AN INTRODUCTION TO GIT

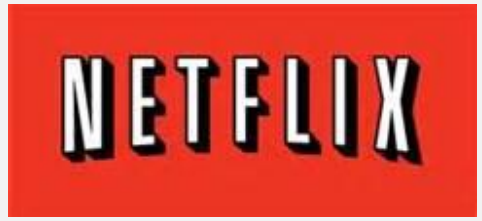
AGENDA

- Introduction
- What is Git?
- Git 101
- Enabling Team Development
- Short vs. Long Lived Branches
- Tools/Resources

HISTORY

Created by Linus Torvalds for work on the Linux kernel ~2005

Some of the companies that use git:



WHAT IS GIT?

GIT IS A

Distributed

Version Control System

GIT IS A

Directory

Content Management System

GIT IS A

Tree

history storage system

DISTRIBUTED

Everyone has the complete history

DISTRIBUTED

Everyone has the complete history

Everything is done offline

...except push/pull

DISTRIBUTED

Everyone has the complete history

Everything is done offline

No central authority

...except by convention

DISTRIBUTED

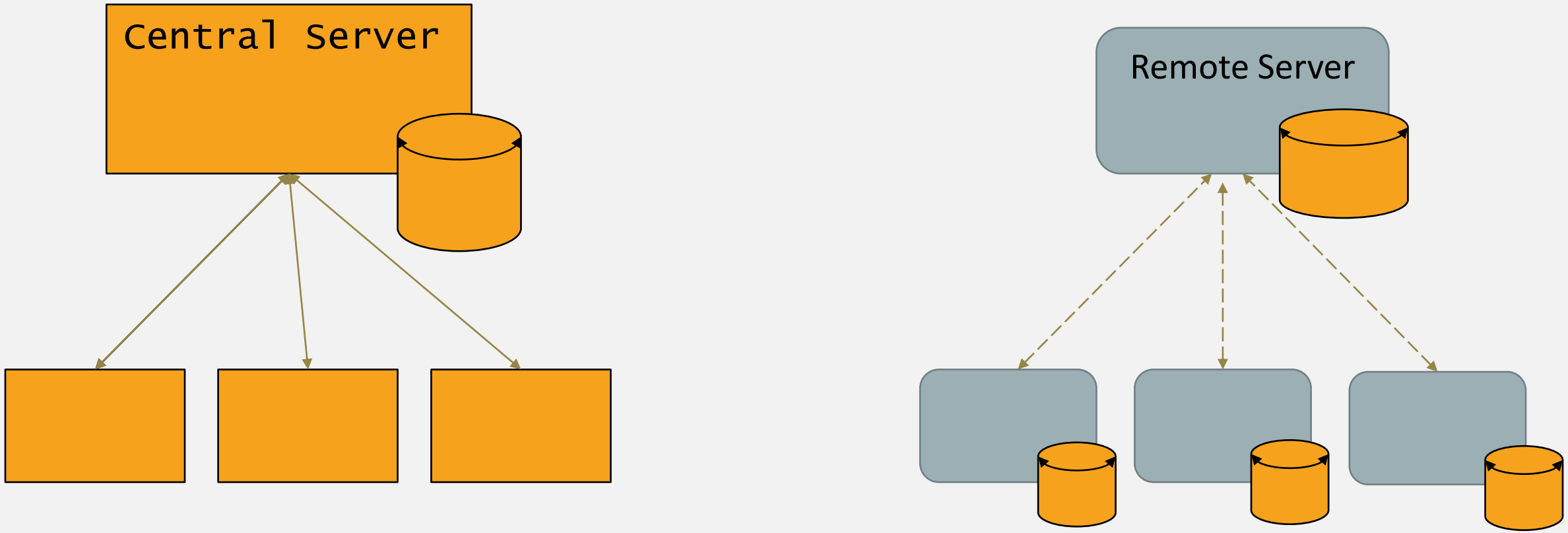
Everyone has the complete history

Everything is done offline

No central authority

Changes can be shared without a server

CENTRALIZED VC VS. DISTRIBUTED VC



BRANCHING

BRANCHING

Forget what you know from Central VC (…TFS, SVN, Perforce...)

BRANCHING

Forget what you know from Central VC

Git branch is “Sticky Note” on a graph node

BRANCHING

Forget what you know from Central VC

Git branch is “Sticky Note” on a graph node

All branch work takes place within the same folder within your file system.

BRANCHING

Forget what you know from Central VC

Git branch is “Sticky Note” on the graph

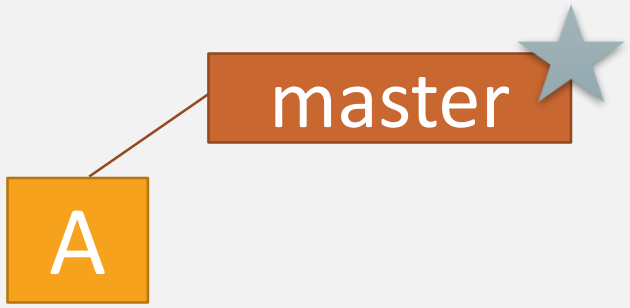
All branch work takes place within the same folder within your file system.

When you switch branches you are moving the “Sticky Note”

INITIALIZATION

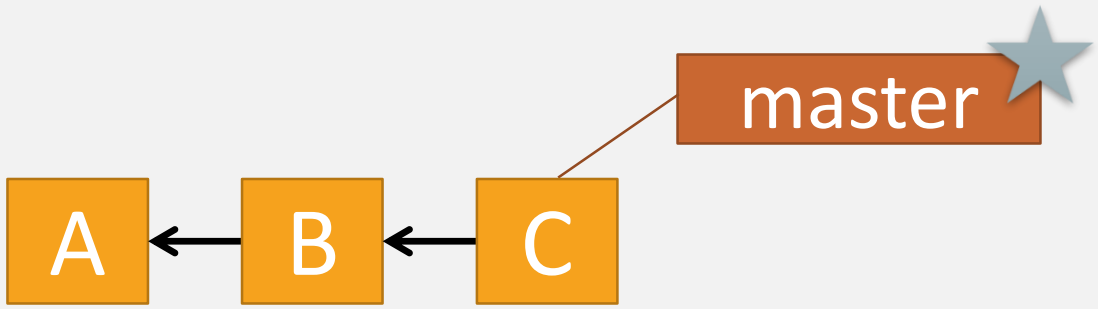
```
C:\> mkdir CoolProject
C:\> cd CoolProject
C:\CoolProject > git init
Initialized empty Git repository in C:/CoolProject/.git
C:\CoolProject > notepad README.txt
C:\CoolProject > git add .
C:\CoolProject > git commit -m 'my first commit'
[master (root-commit) 7106a52] my first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.txt
```

BRANCHES ILLUSTRATED



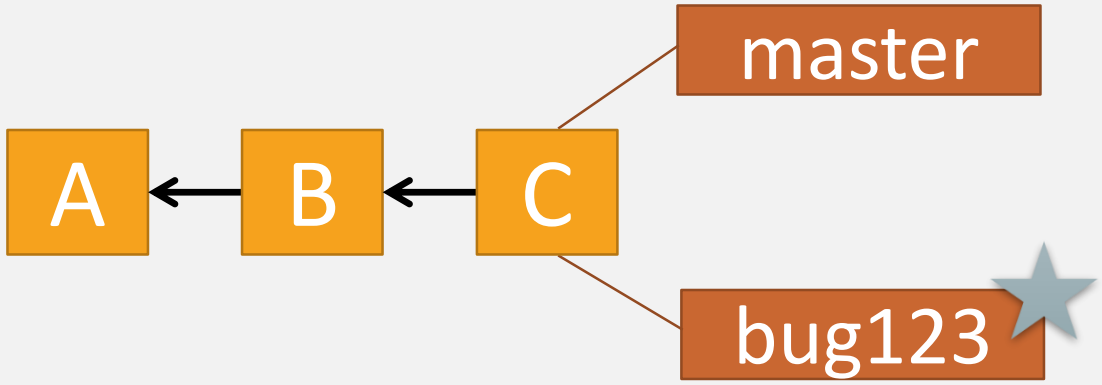

```
> git commit –m ‘my first commit’
```

BRANCHES ILLUSTRATED



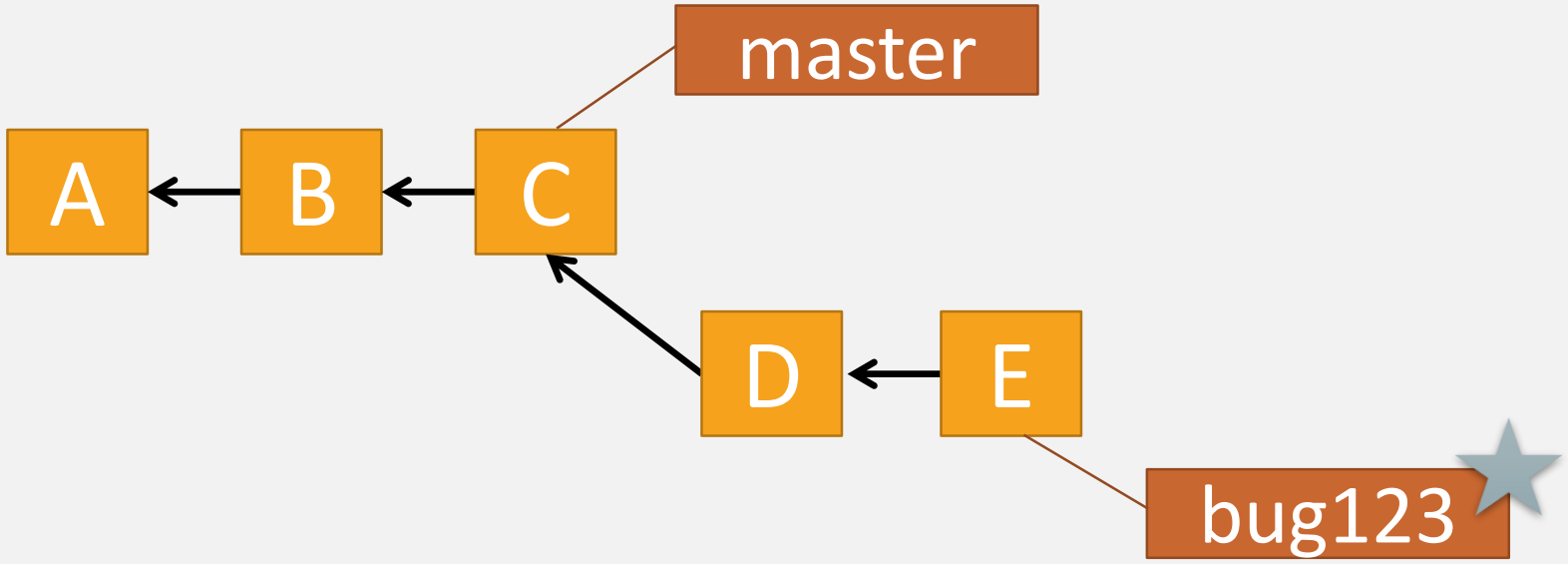

```
> git commit (x2)
```

BRANCHES ILLUSTRATED



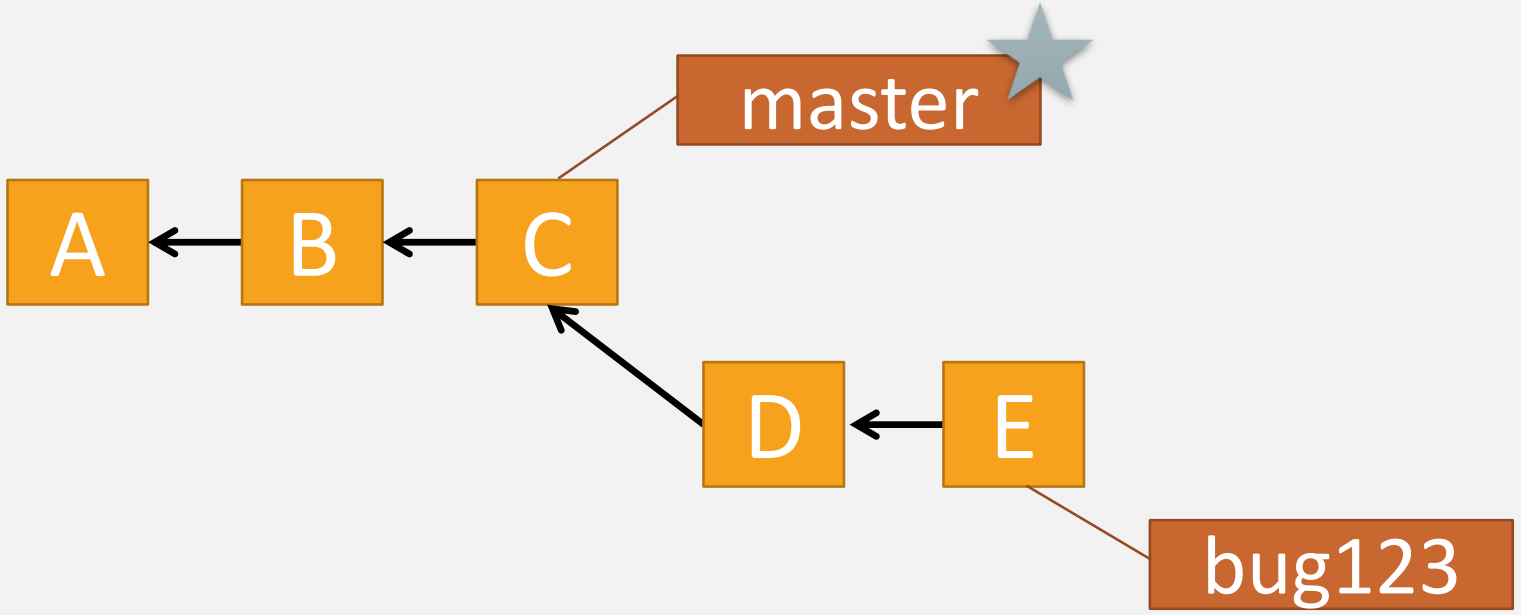

```
> git checkout -b bug123
```

BRANCHES ILLUSTRATED



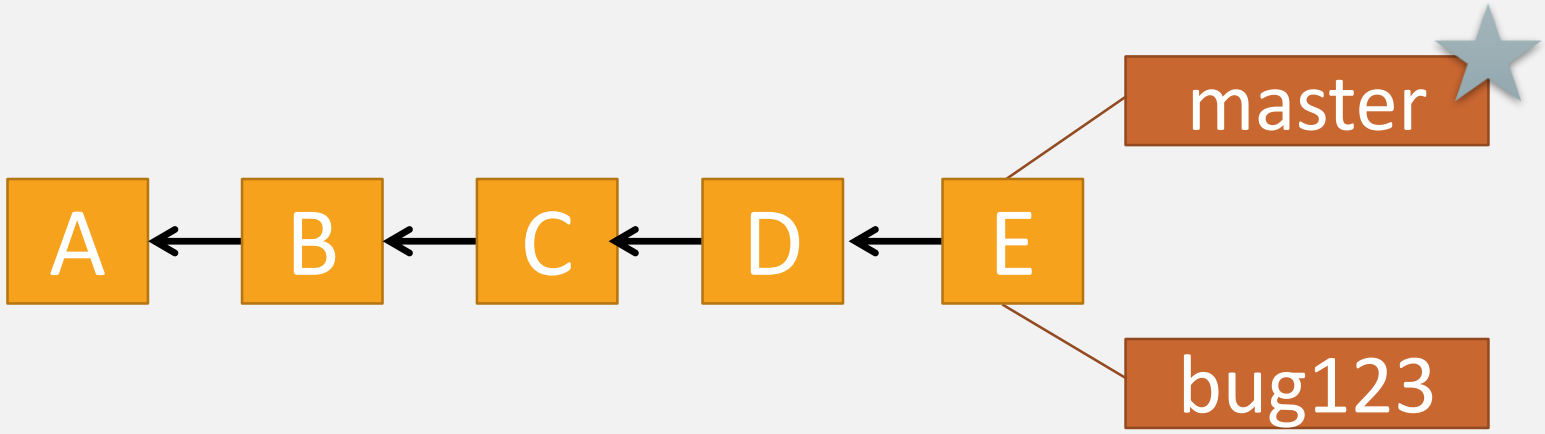

```
> git commit (x2)
```

BRANCHES ILLUSTRATED



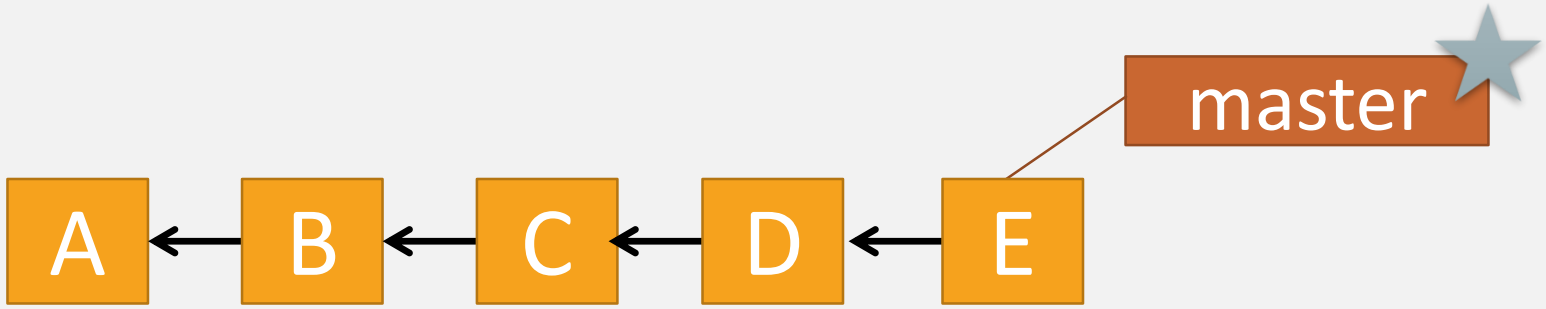

```
> git checkout master
```

BRANCHES ILLUSTRATED



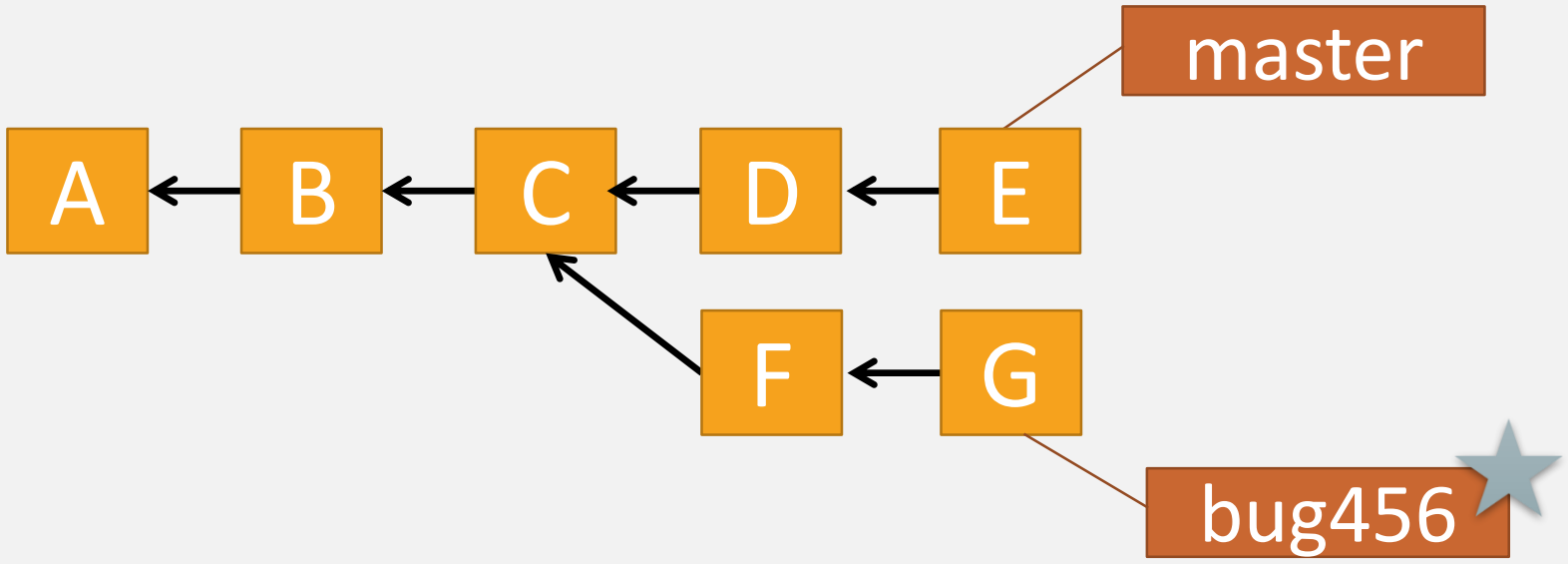

```
> git merge bug123
```

BRANCHES ILLUSTRATED

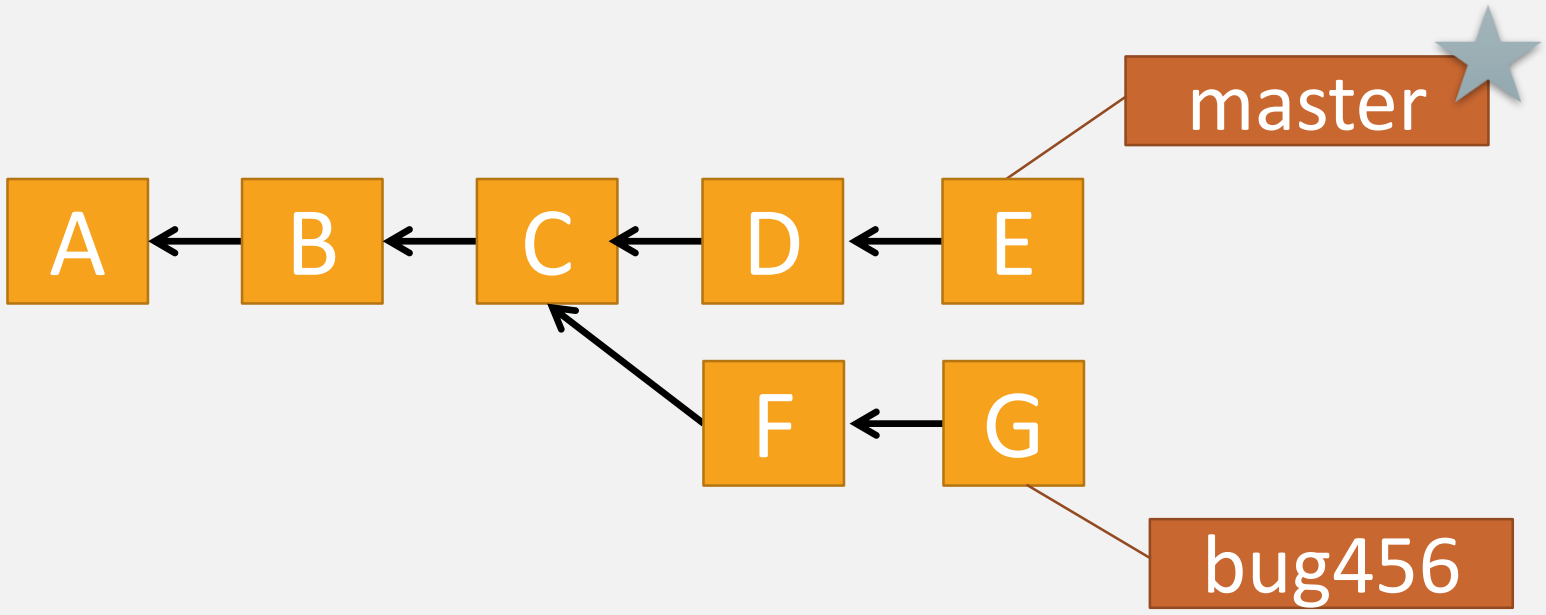



```
> git branch -d bug123
```

BRANCHES ILLUSTRATED

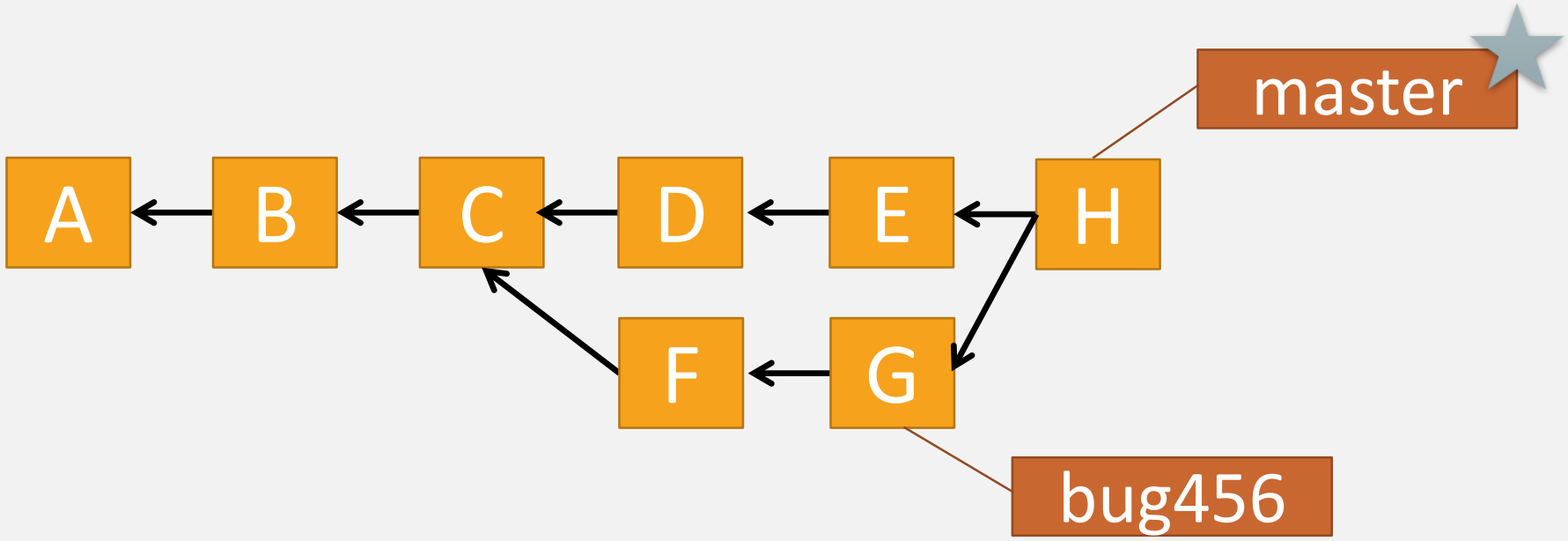


BRANCHES ILLUSTRATED



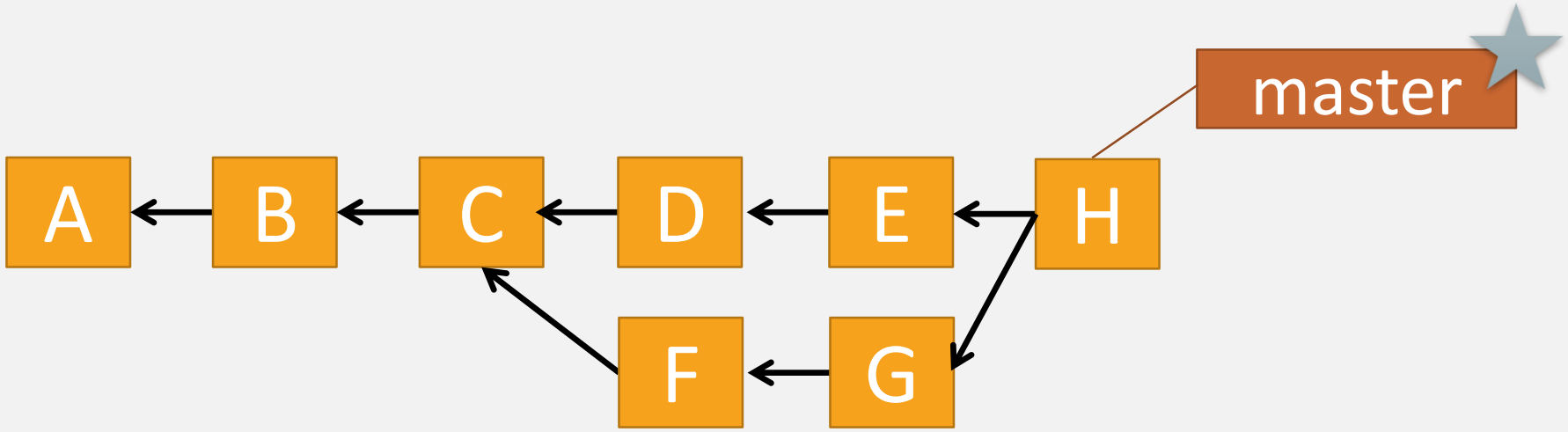

```
> git checkout master
```

BRANCHES ILLUSTRATED



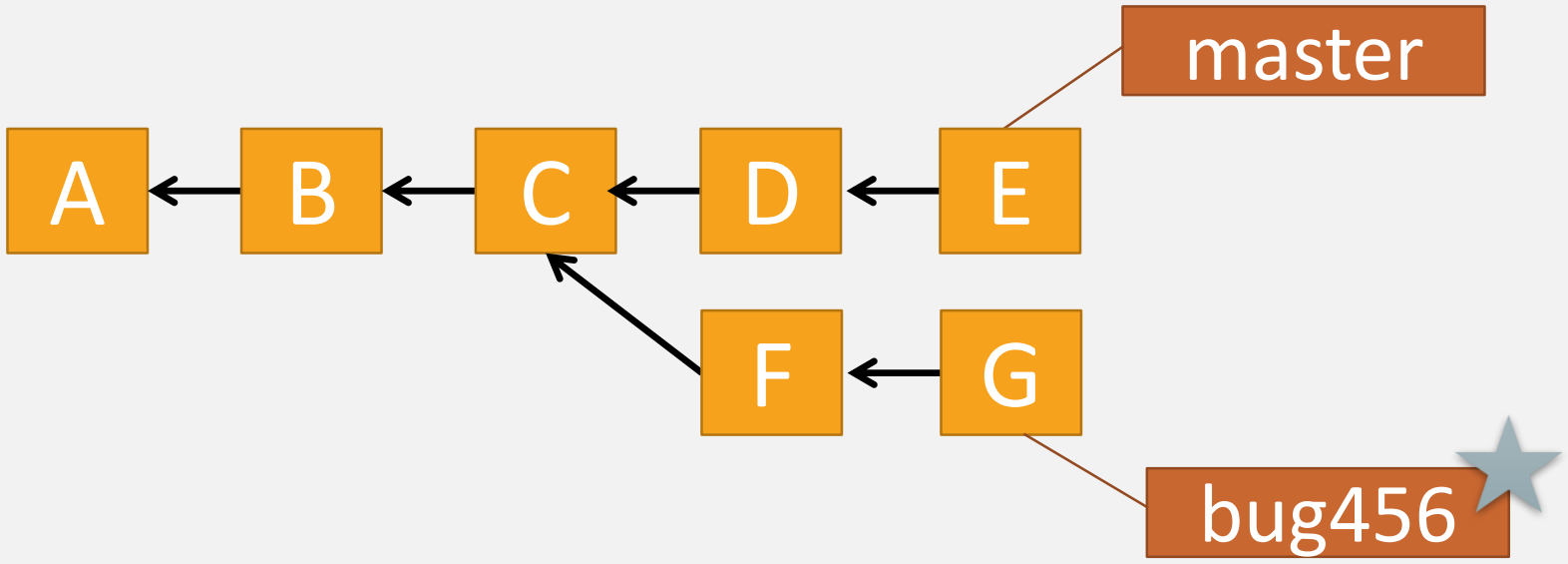

```
> git merge bug456
```

BRANCHES ILLUSTRATED

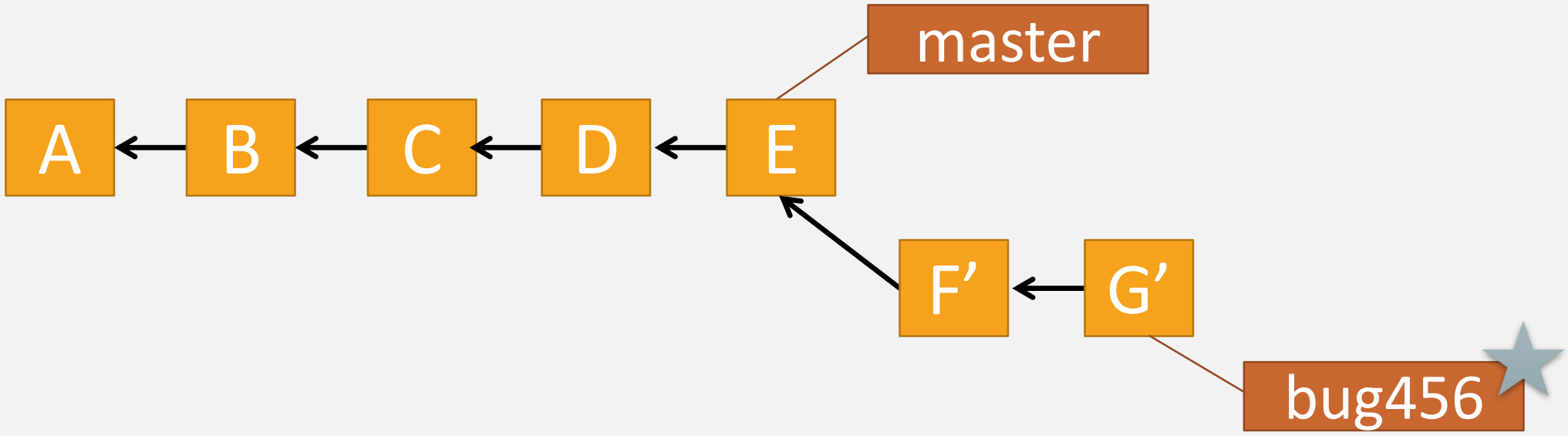



```
> git branch -d bug456
```

BRANCHES ILLUSTRATED

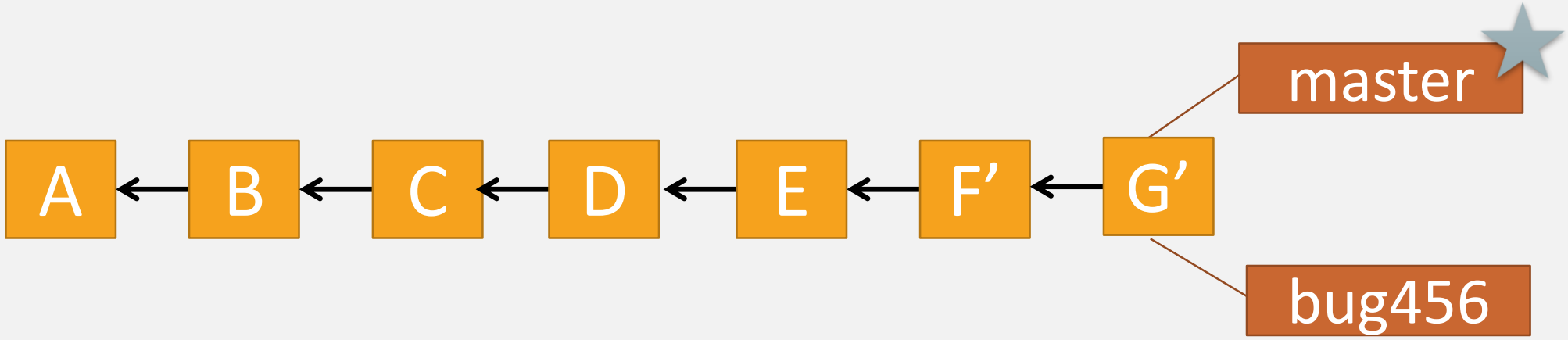


BRANCHES ILLUSTRATED




```
> git rebase master
```

BRANCHES ILLUSTRATED




```
> git checkout master
> git merge bug456
```

BRANCHING REVIEW

BRANCHING REVIEW

Quick and Easy to create ‘Feature’ Branches

BRANCHING REVIEW

Quick and Easy to create ‘Feature’ Branches

Local branches are very powerful

BRANCHING REVIEW

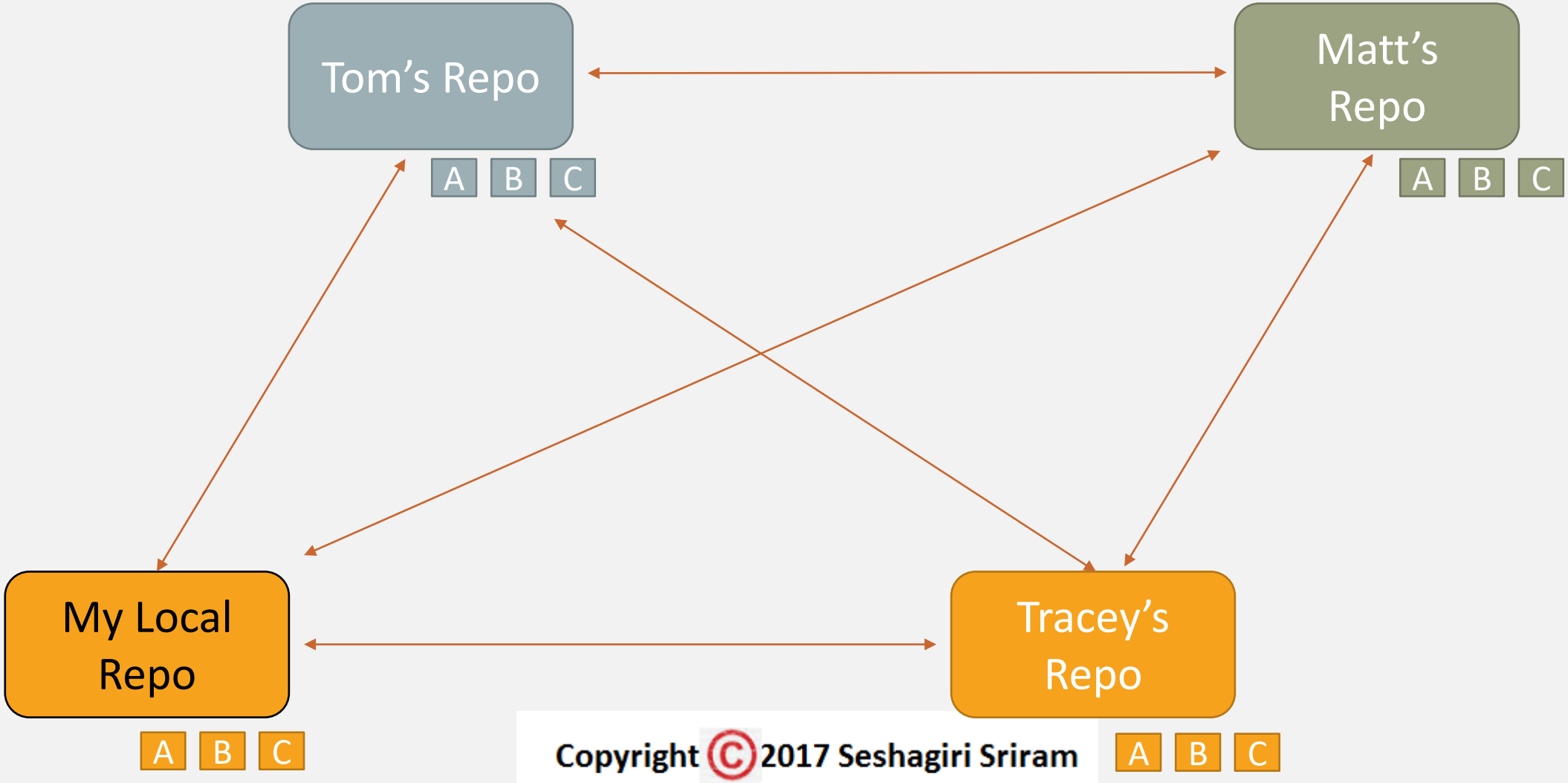
Quick and Easy to create ‘Feature’ Branches

Local branches are very powerful

Rebase is not scary

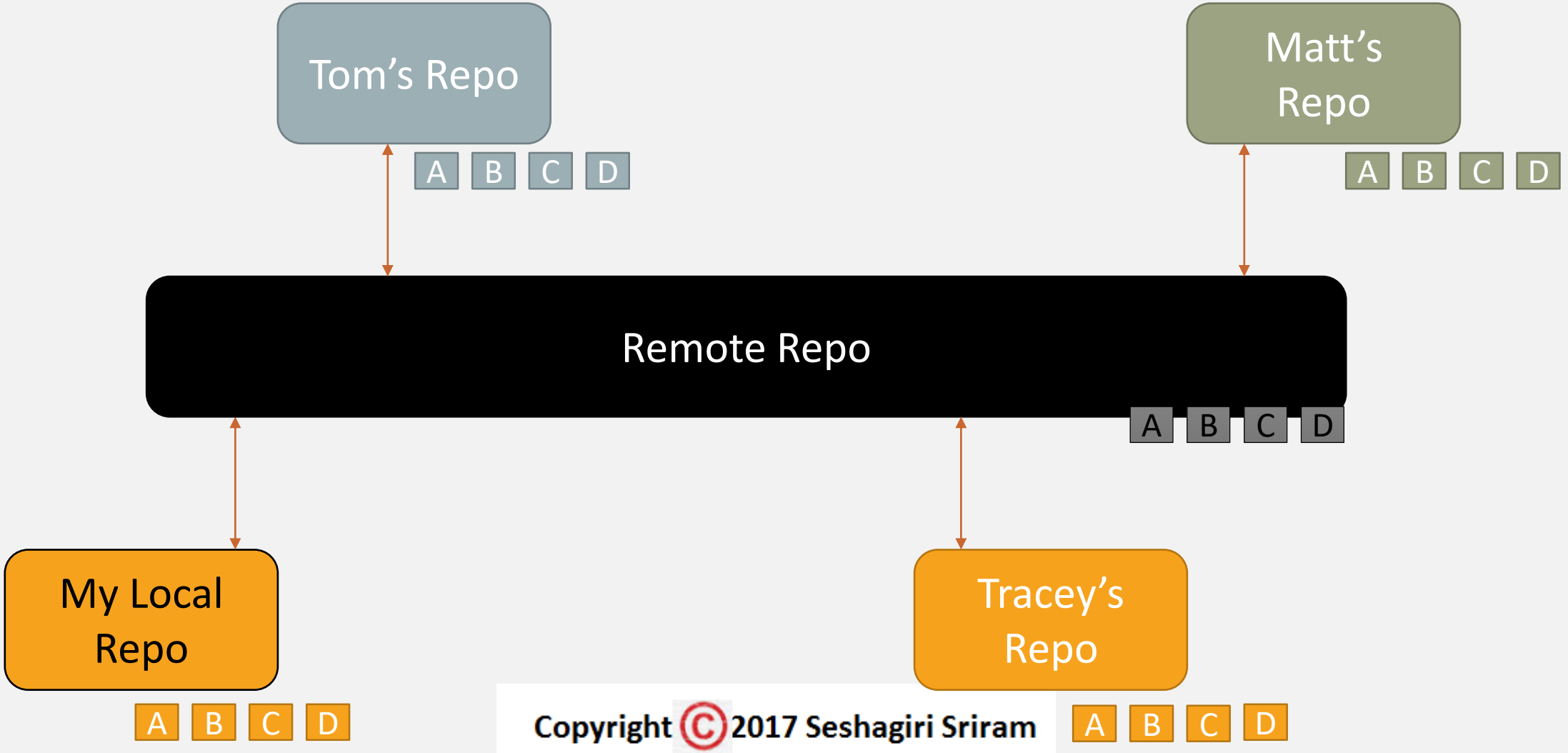
SOFTWARE IS A TEAM SPORT

SHARING COMMITS



ADDING A REMOTE

SHARING COMMITS



SETTING UP A REMOTE

SETTING UP A REMOTE

Adding a remote to an existing local repo

```
C:\CoolProject > git remote add origin
https://git01.codeplex.com/coolproject
C:\CoolProject > git remote -v
origin  https://git01.codeplex.com/coolproject (fetch)
origin  https://git01.codeplex.com/coolproject (push)
```

SETTING UP A REMOTE

Clone will auto setup the remote

```
C:\> git clone https://git01.codeplex.com/coolproject
Cloning into 'coolproject'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
C:\> cd .\coolproject
C:\CoolProject> git remote -v
origin  https://git01.codeplex.com/coolproject (fetch)
origin  https://git01.codeplex.com/coolproject (push)
```

SETTING UP A REMOTE

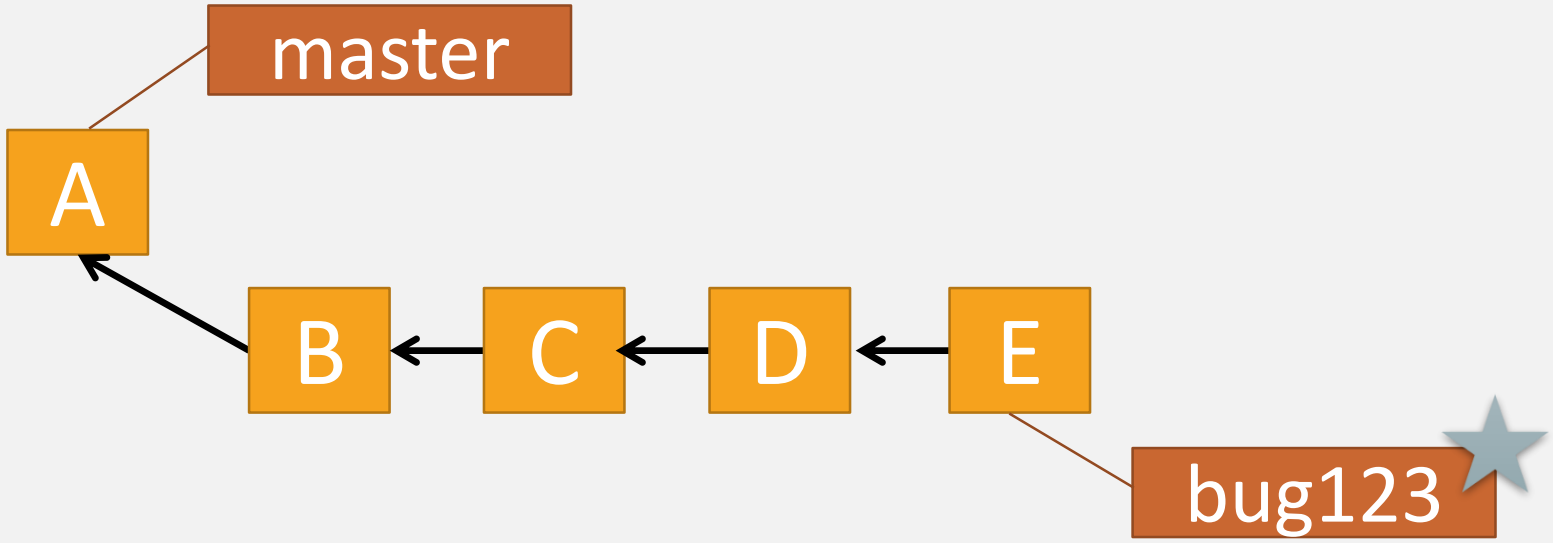
Name remotes what you want

SETTING UP A REMOTE

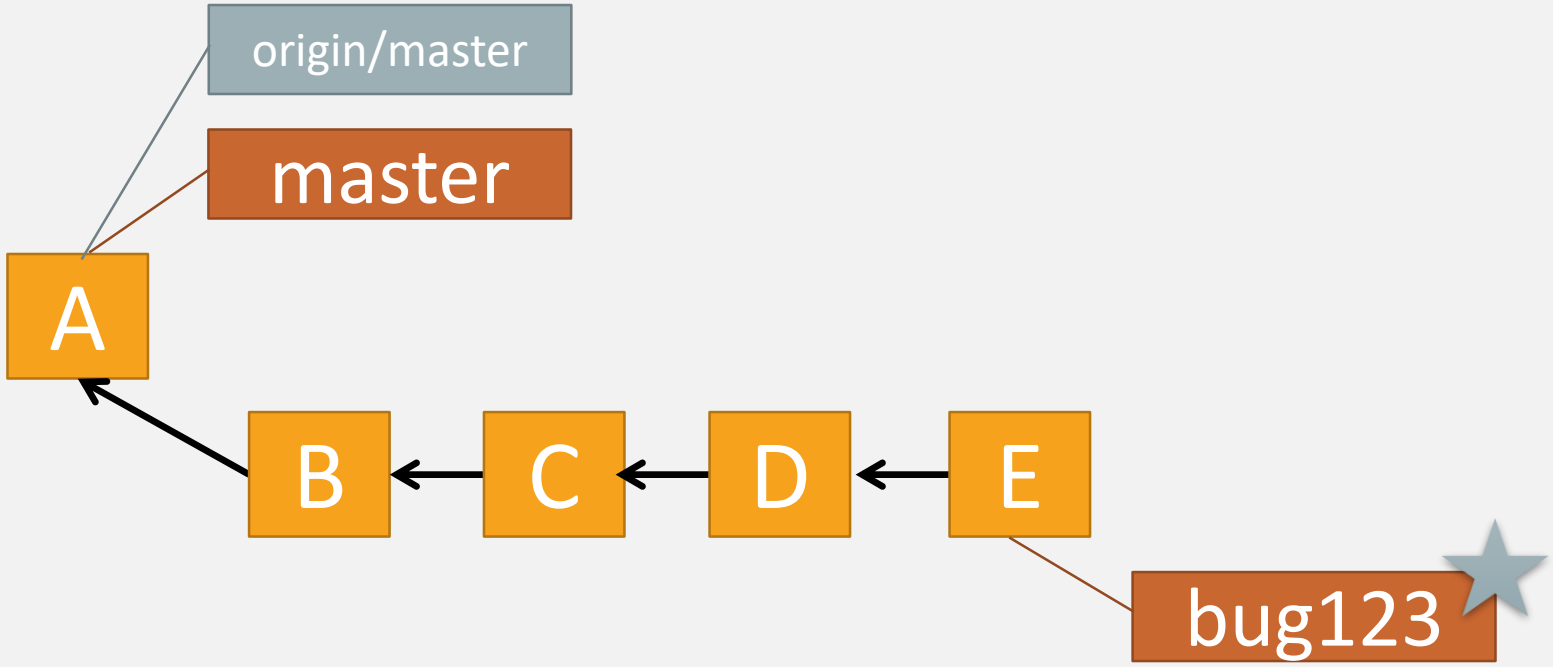
Name remotes what you want

Origin is only a convention

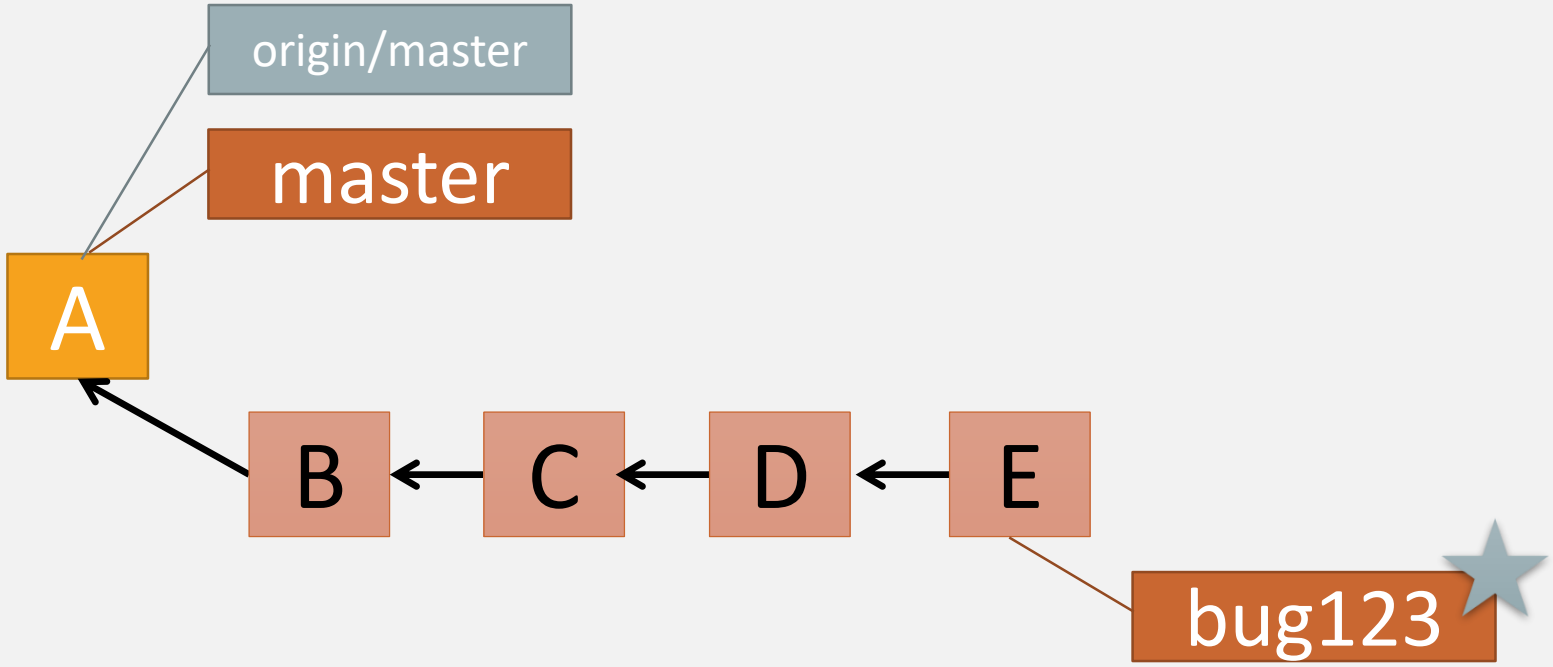
BRANCHES ILLUSTRATED



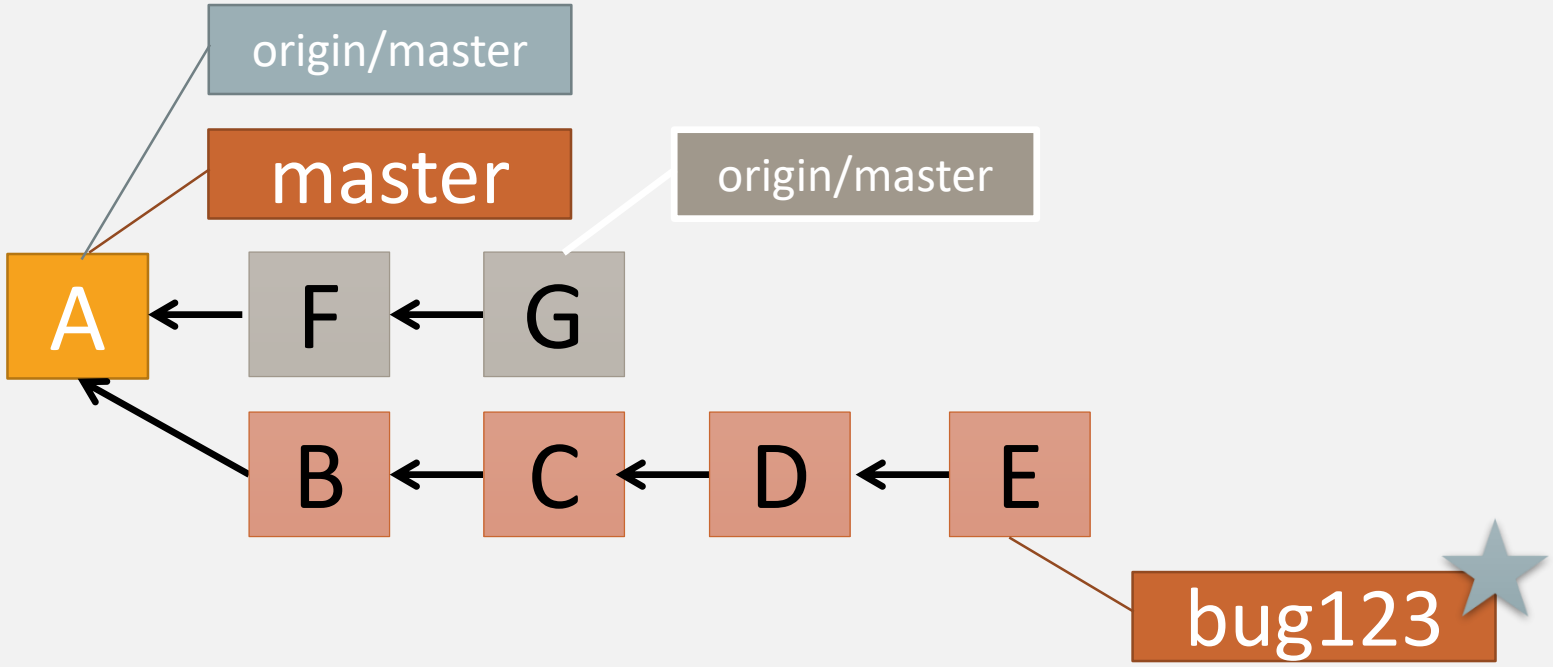
BRANCHES ILLUSTRATED



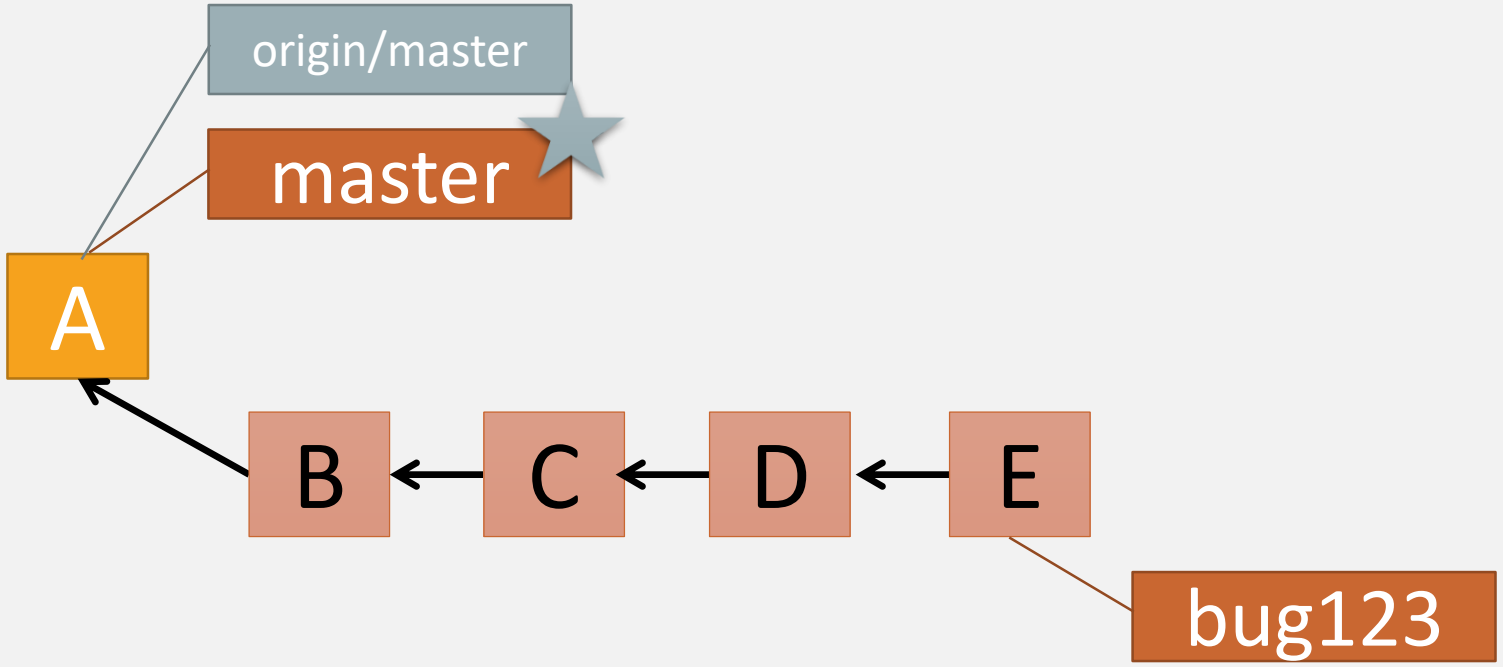
BRANCHES ILLUSTRATED



BRANCHES ILLUSTRATED

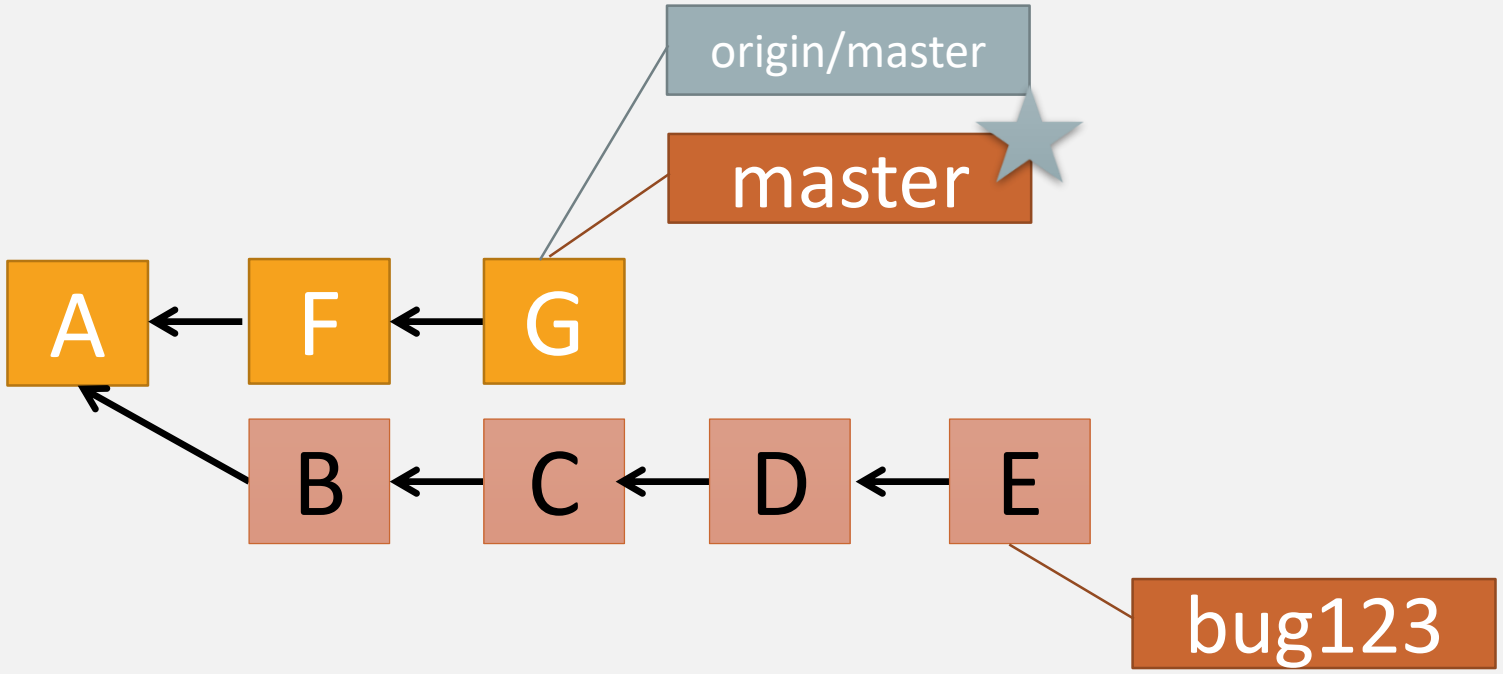


BRANCHES ILLUSTRATED




```
> git checkout master
```

BRANCHES ILLUSTRATED



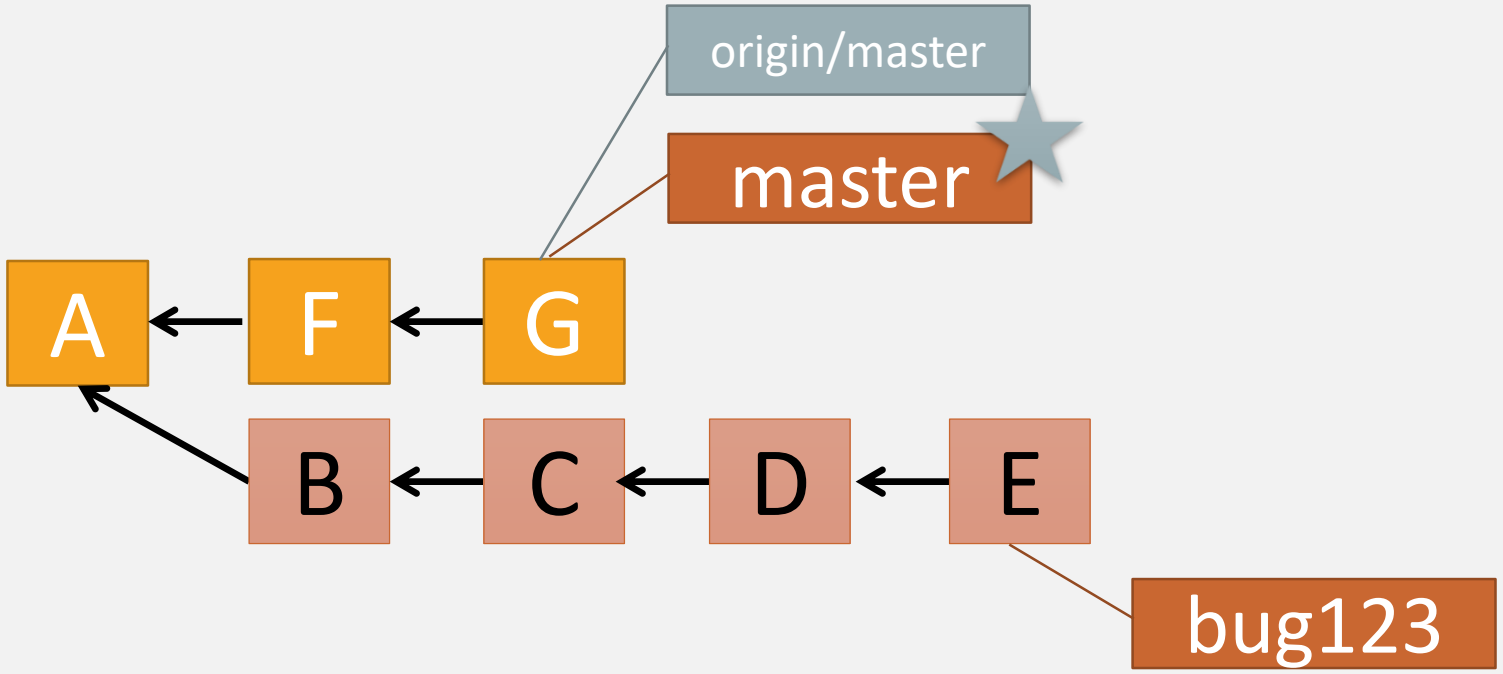

```
> git pull origin
```

Pull = Fetch + Merge

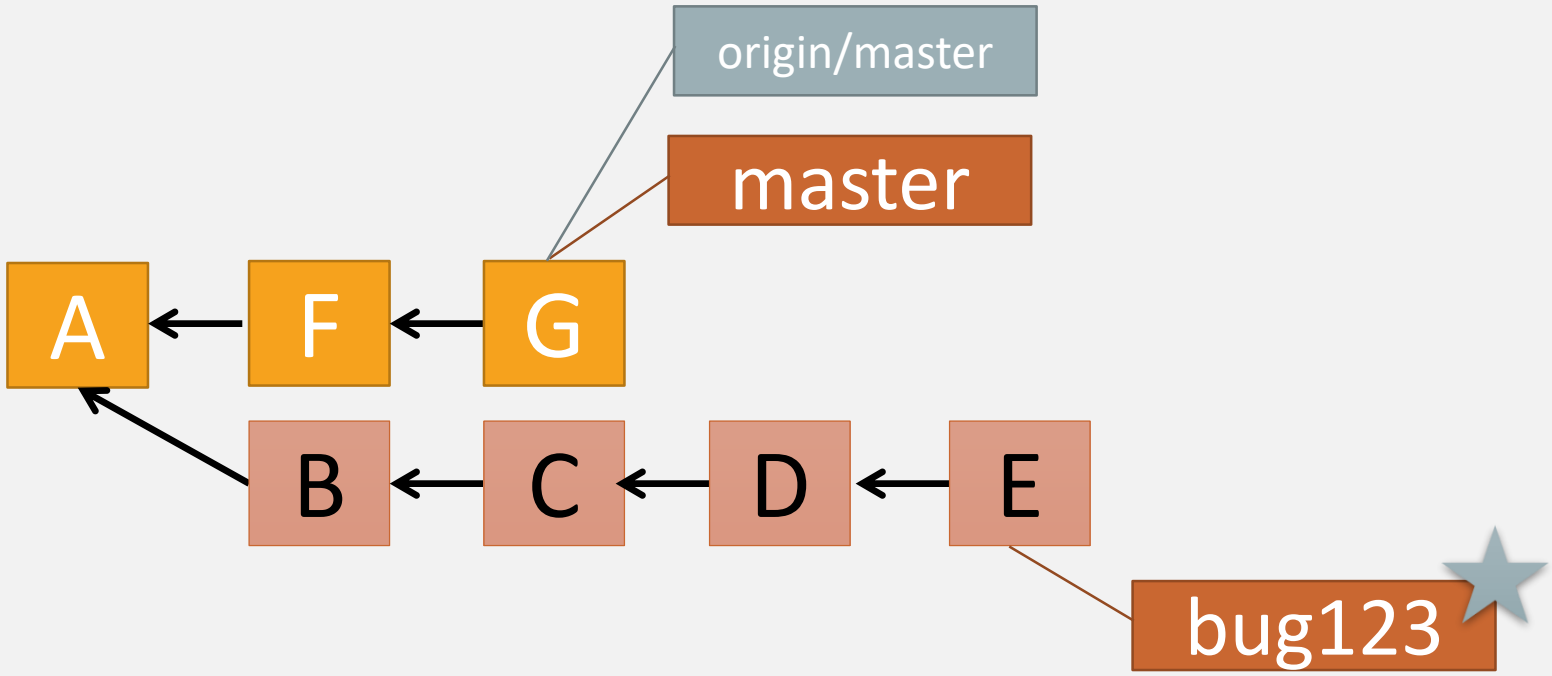
Fetch - updates your local copy of the remote branch

Pull essentially does a fetch and then runs the merge in one step.

BRANCHES ILLUSTRATED

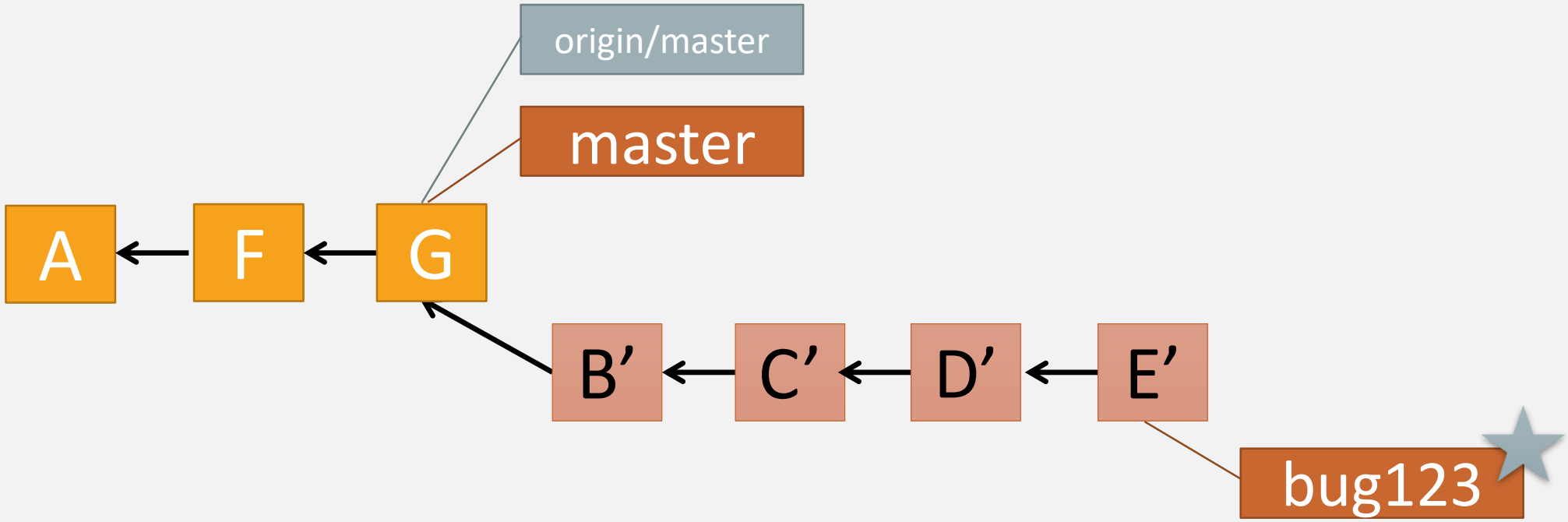


BRANCHES ILLUSTRATED



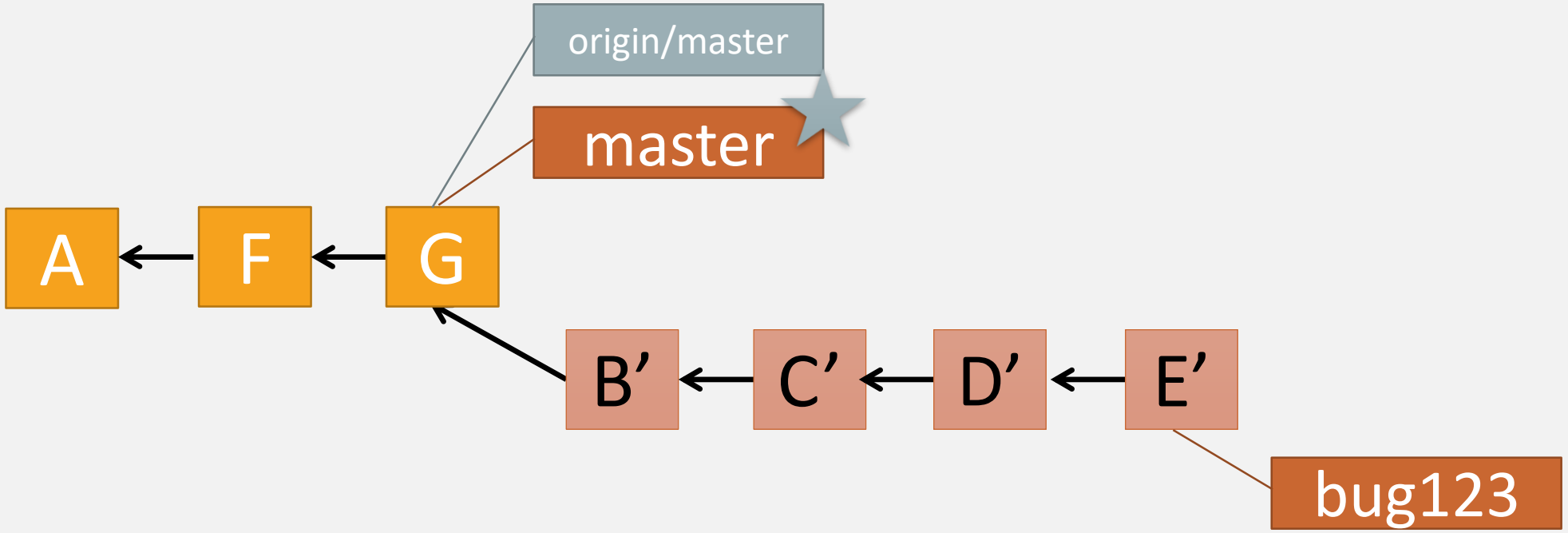

```
> git checkout bug123
```

BRANCHES ILLUSTRATED



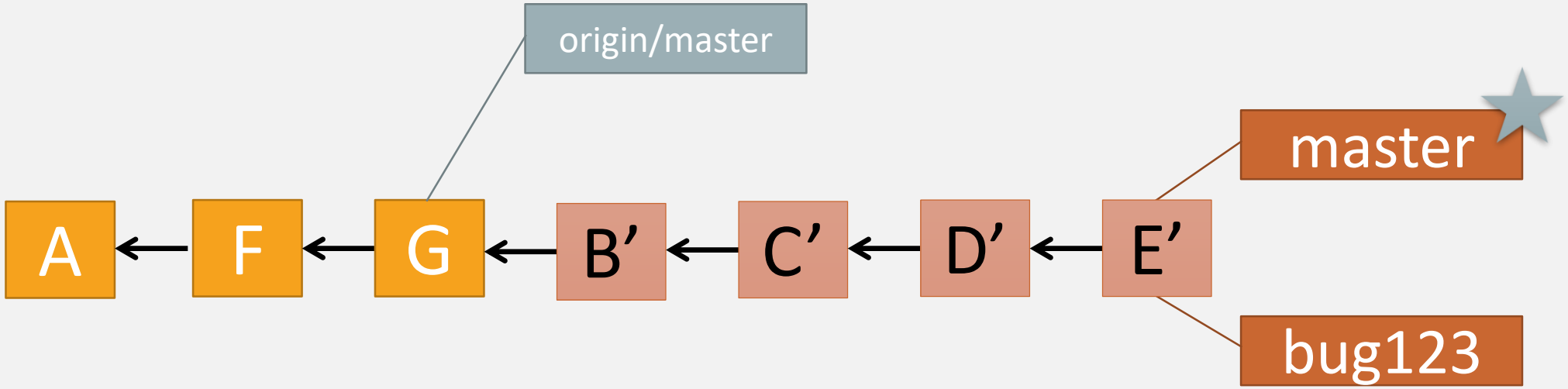

```
> git rebase master
```

BRANCHES ILLUSTRATED



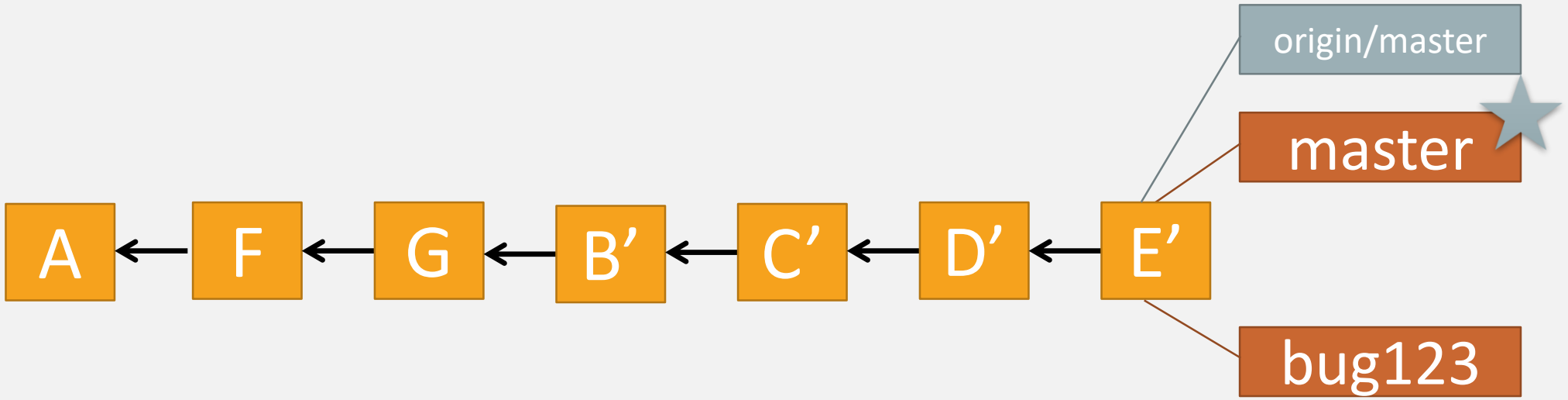

```
> git checkout master
```

BRANCHES ILLUSTRATED




```
> git merge bug123
```

BRANCHES ILLUSTRATED




```
> git push origin
```

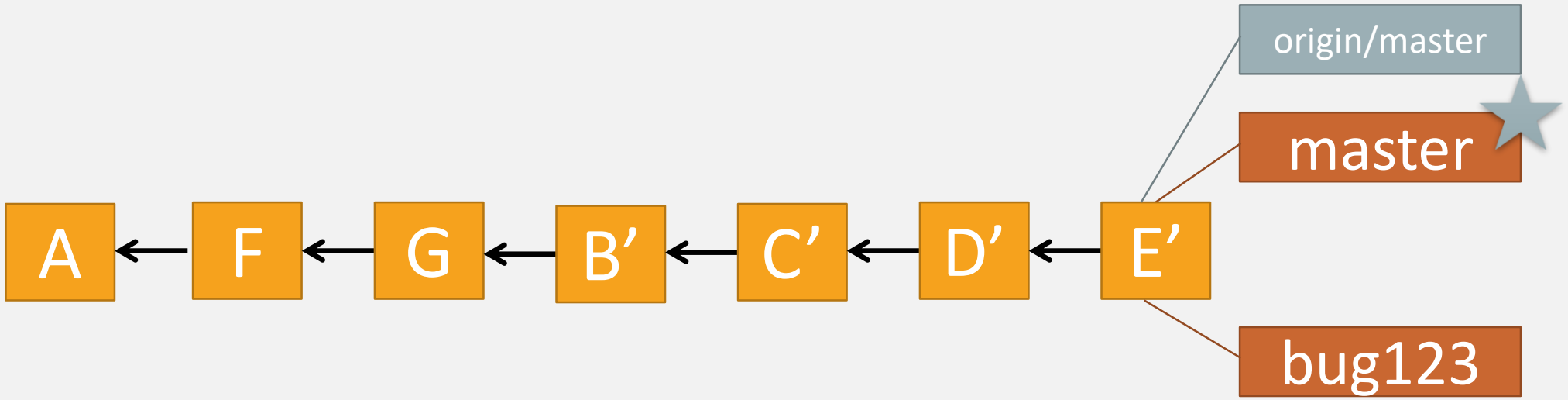
Push

Pushes your changes upstream

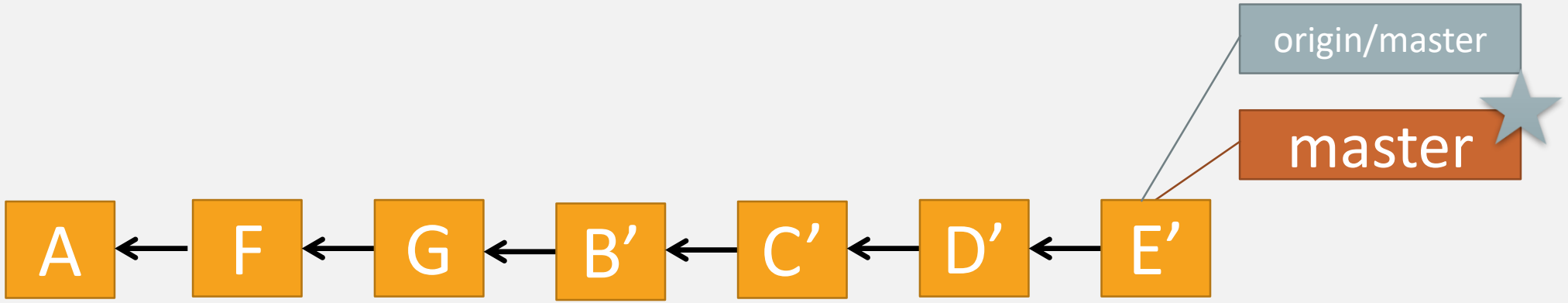
Git will reject pushes if newer changes exist on remote.

Good practice: Pull then Push

BRANCHES ILLUSTRATED



BRANCHES ILLUSTRATED




```
> git branch -d bug123
```

Adding a Remote Review

Adding a remote makes it easy to share

Pulling from the remote often helps keep you up to date

SHORT VS. LONG-LIVED BRANCHES

Local branches are short lived

SHORT VS. LONG-LIVED BRANCHES

Local branches are short lived

Staying off master keeps merges simple

SHORT VS. LONG-LIVED BRANCHES

Local branches are short lived

Staying off master keeps merges simple

Enables working on several changes at once

SHORT VS. LONG-LIVED BRANCHES

Local branches are short lived

Staying off master keeps merges simple

Enables working on several changes at once

Create **Commit** **Merge** **Delete**

SHORT VS. LONG-LIVED BRANCHES

Great for multi-version work

SHORT VS. LONG-LIVED BRANCHES

Great for multi-version work

Follow same rules as Master

SHORT VS. LONG-LIVED BRANCHES

Great for multi-version work

Follow same rules as Master…Story branches

SHORT VS. LONG-LIVED BRANCHES

Great for multi-version work

Follow same rules as Master…Story branches

Integrate frequently

SHORT VS. LONG-LIVED BRANCHES

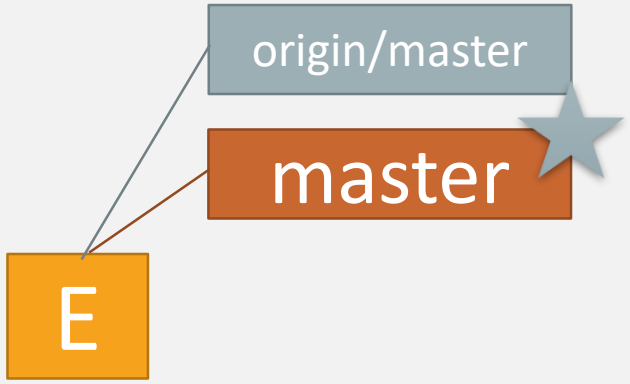
Great for multi-version work

Follow same rules as Master…Story branches

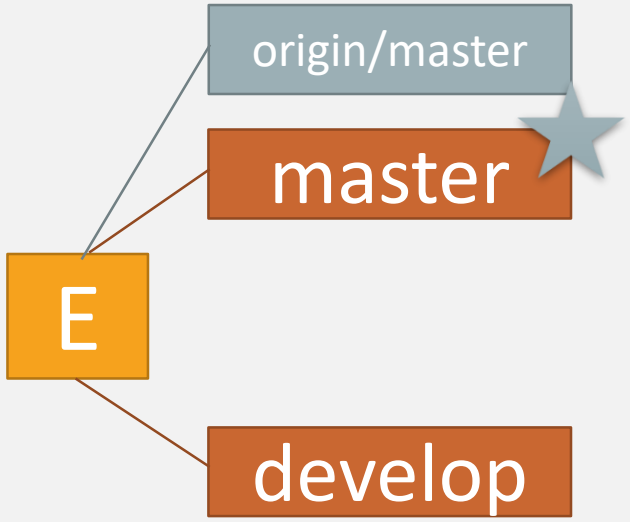
Integrate frequently

Pushed to Remotes

BRANCHES ILLUSTRATED

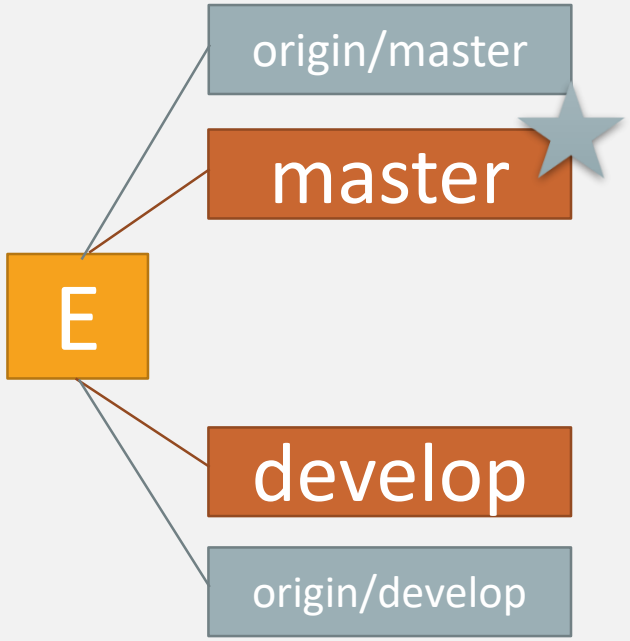


BRANCHES ILLUSTRATED



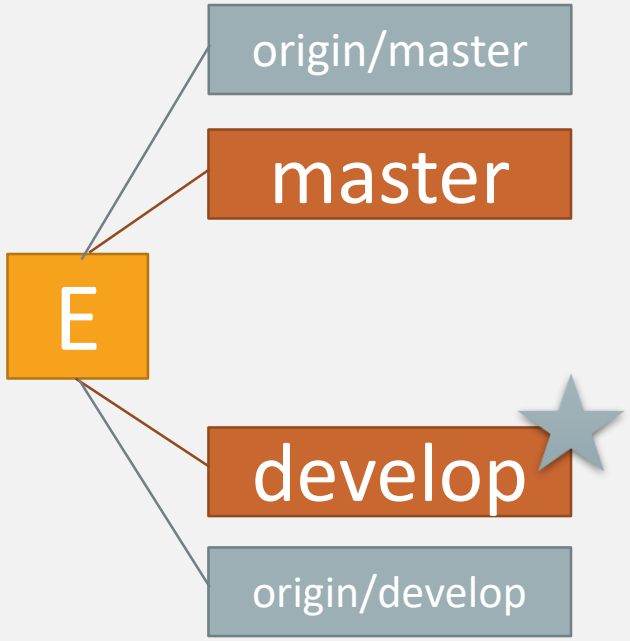

```
> git branch develop
```

BRANCHES ILLUSTRATED



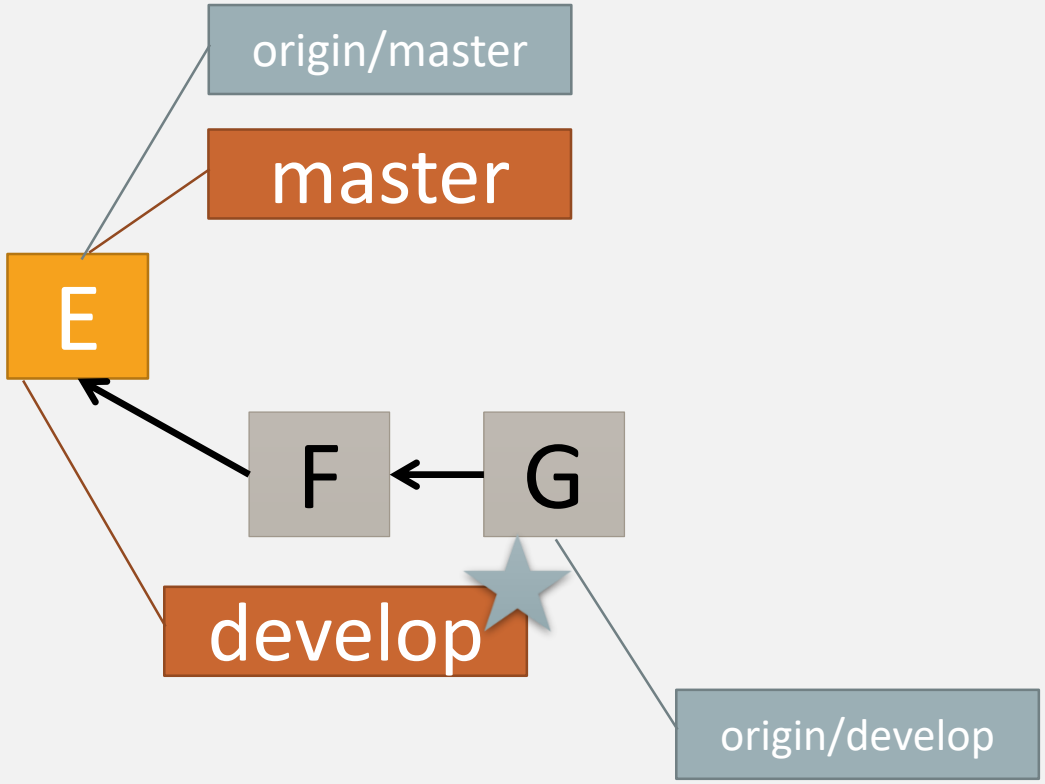

```
> git push origin develop
```

BRANCHES ILLUSTRATED

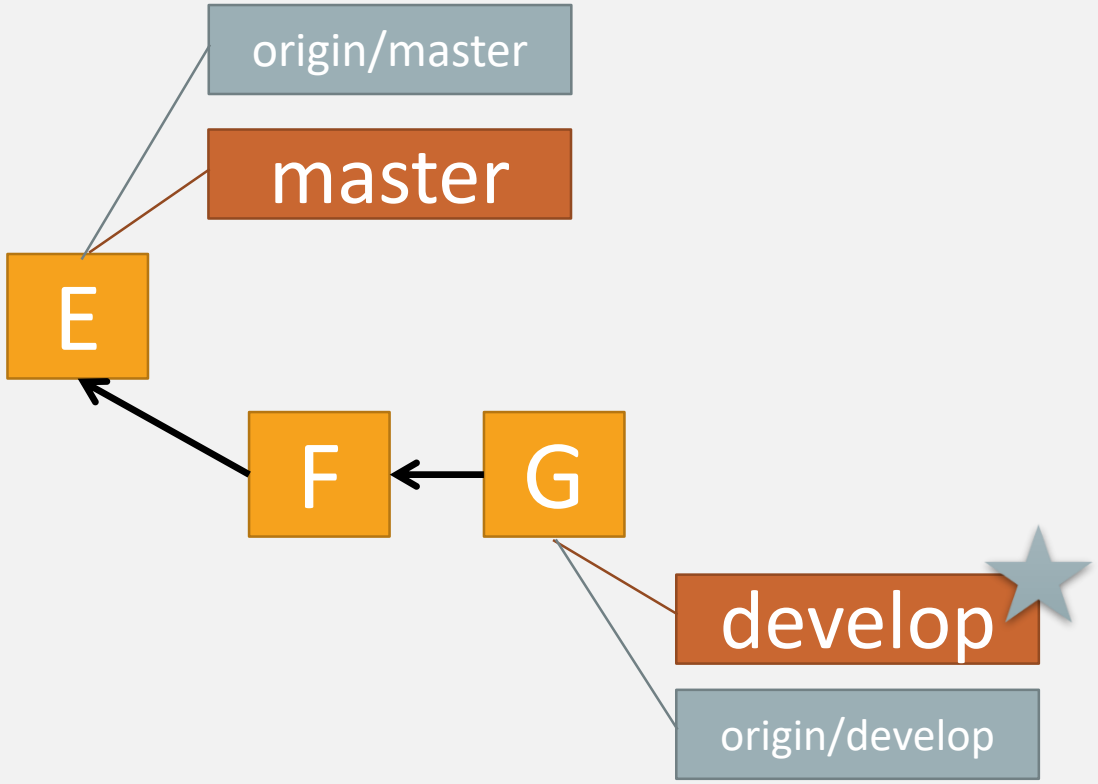



```
> git checkout develop
```

BRANCHES ILLUSTRATED

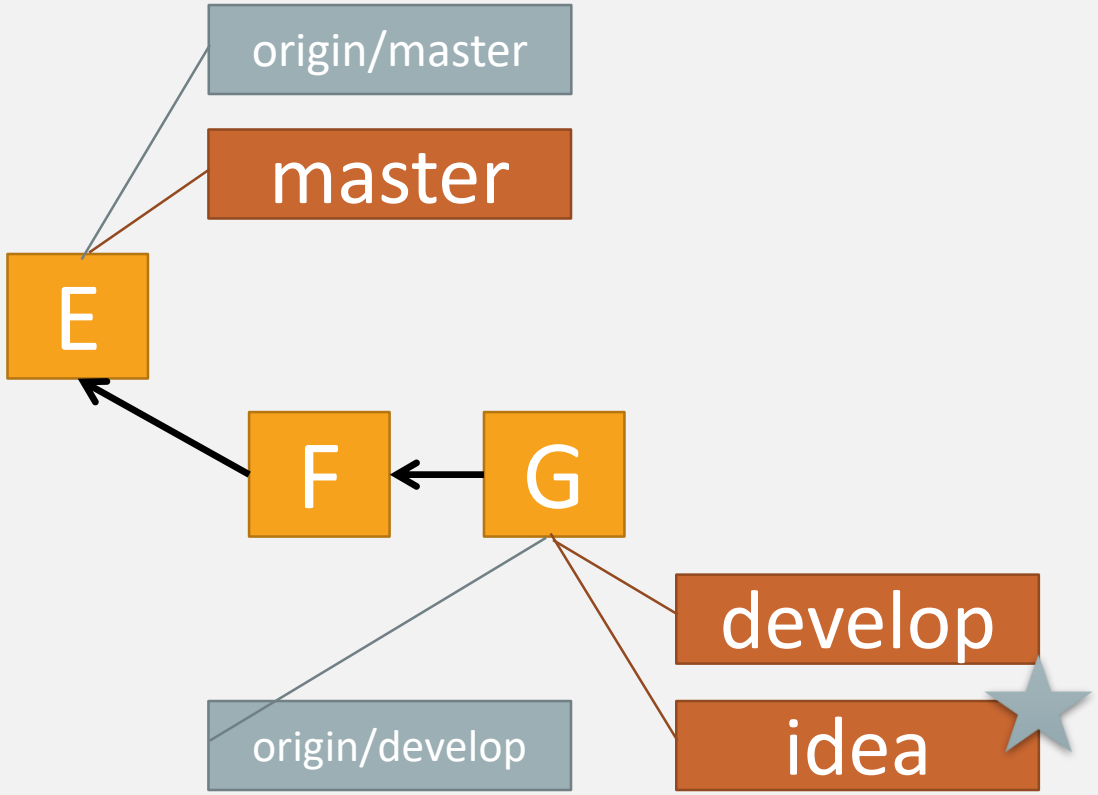


BRANCHES ILLUSTRATED



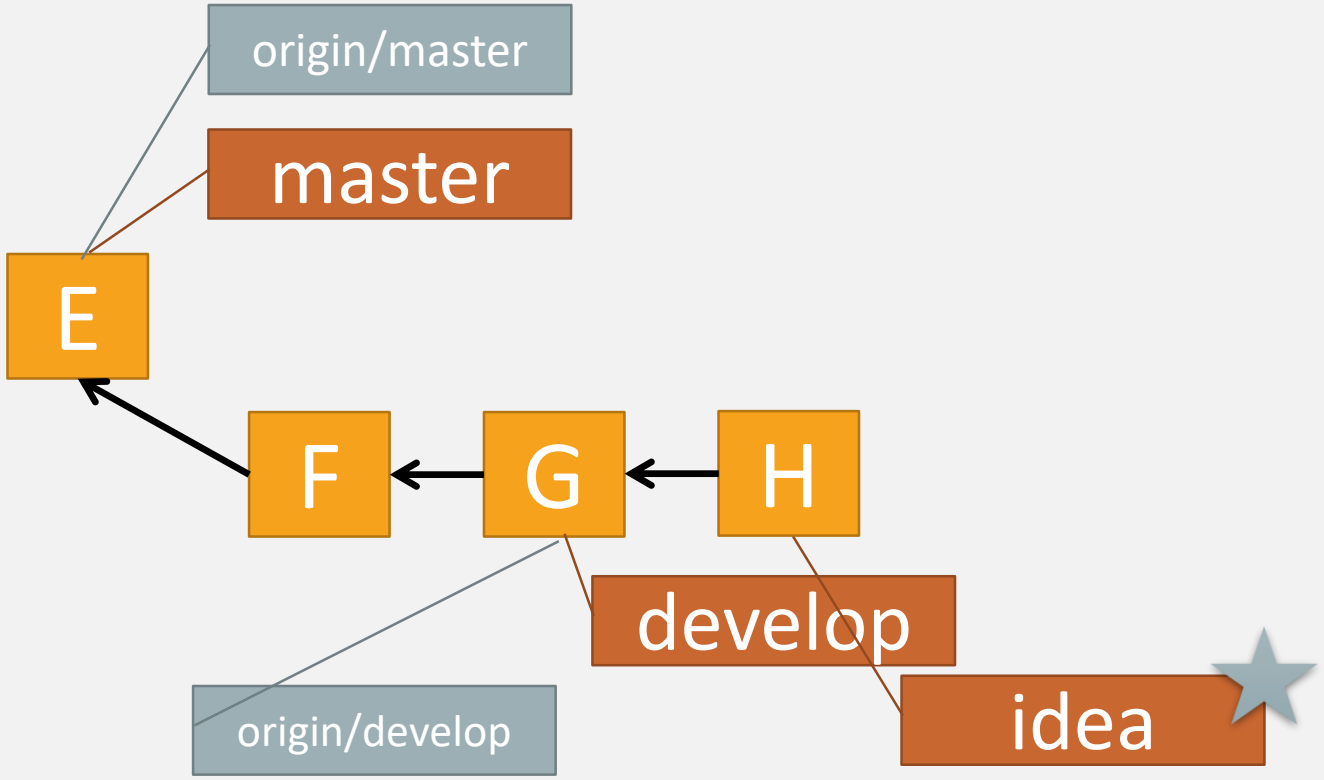

```
> git pull origin develop
```

BRANCHES ILLUSTRATED



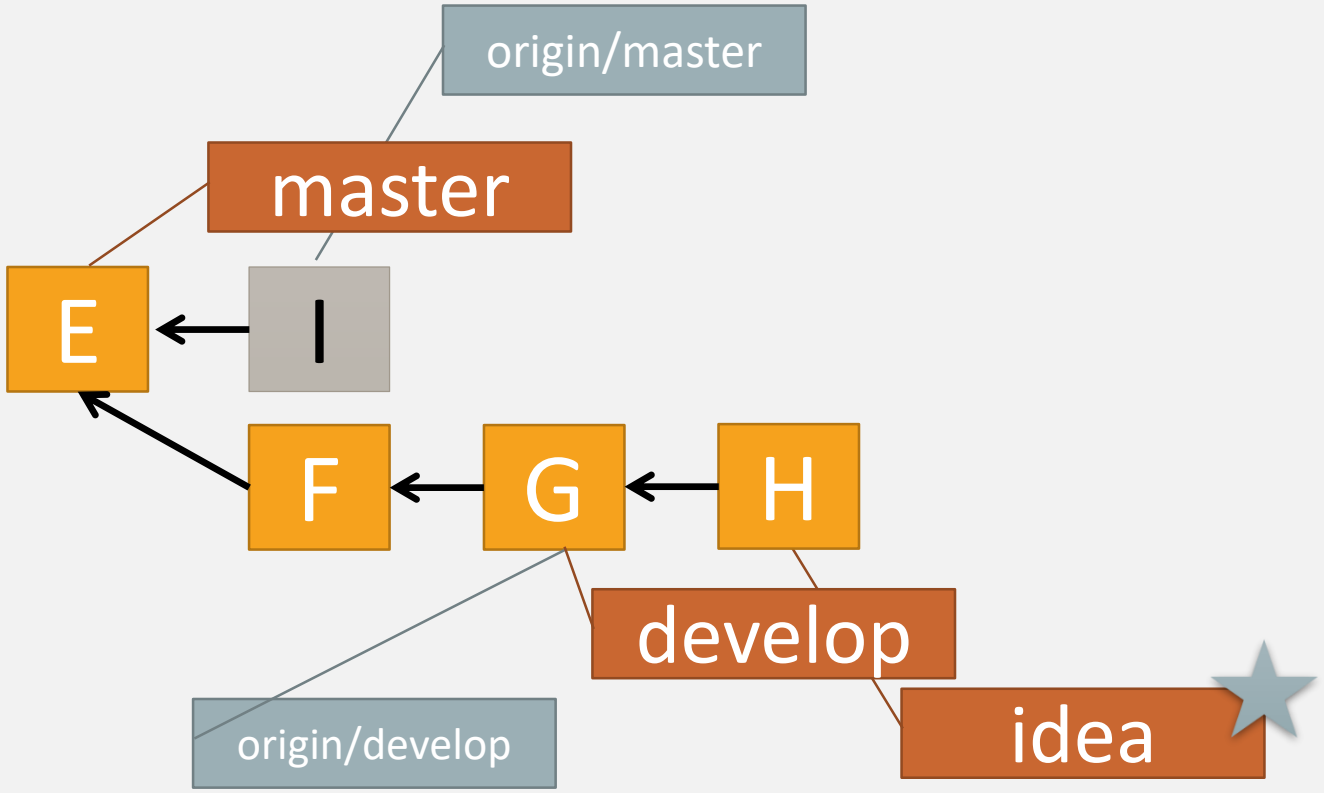

```
> git checkout -b idea
```

BRANCHES ILLUSTRATED

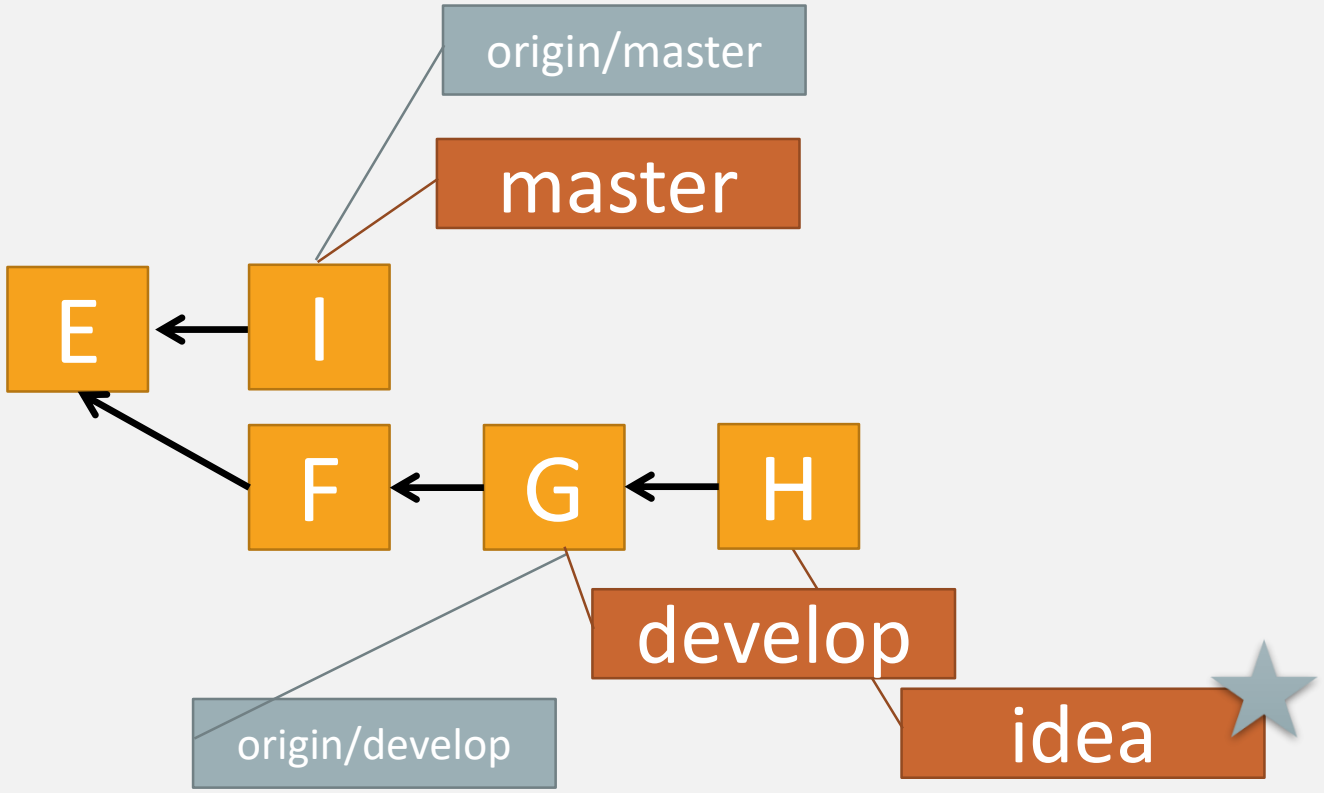



```
> git commit
```

BRANCHES ILLUSTRATED

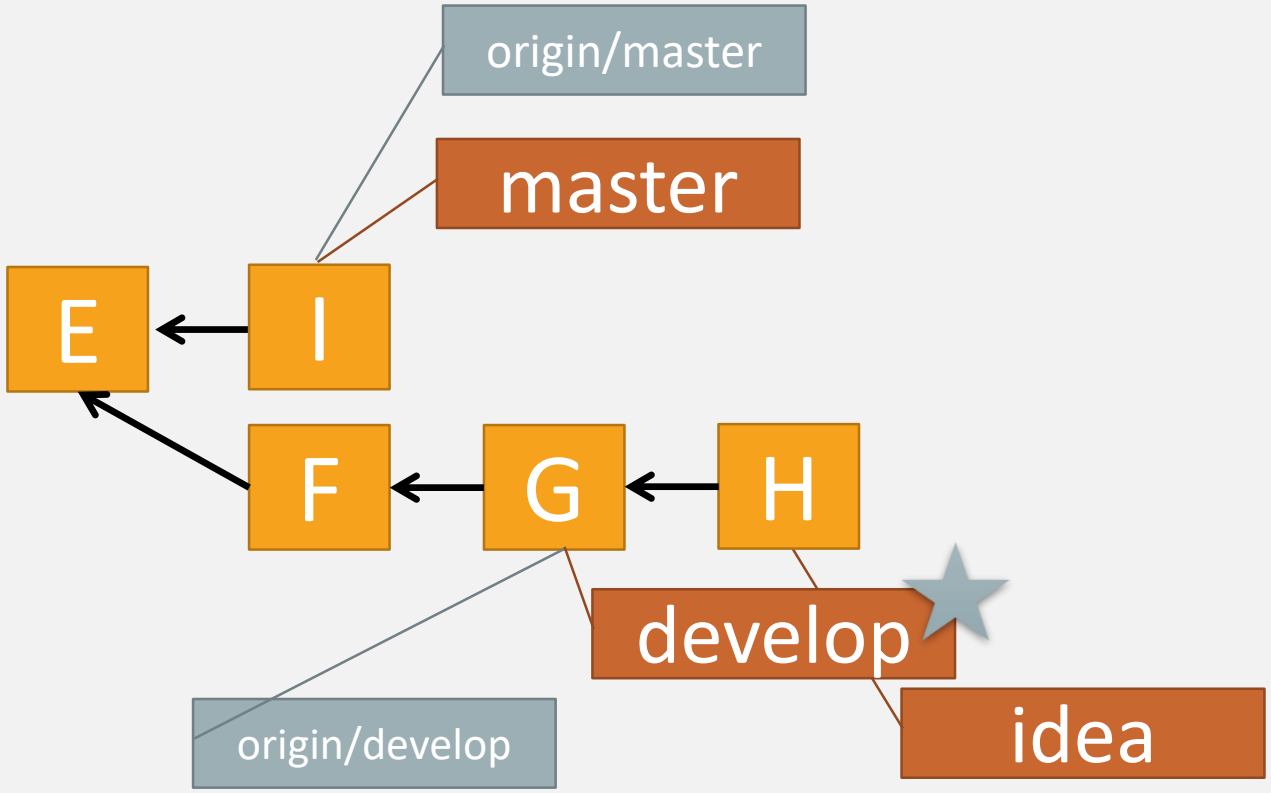


BRANCHES ILLUSTRATED



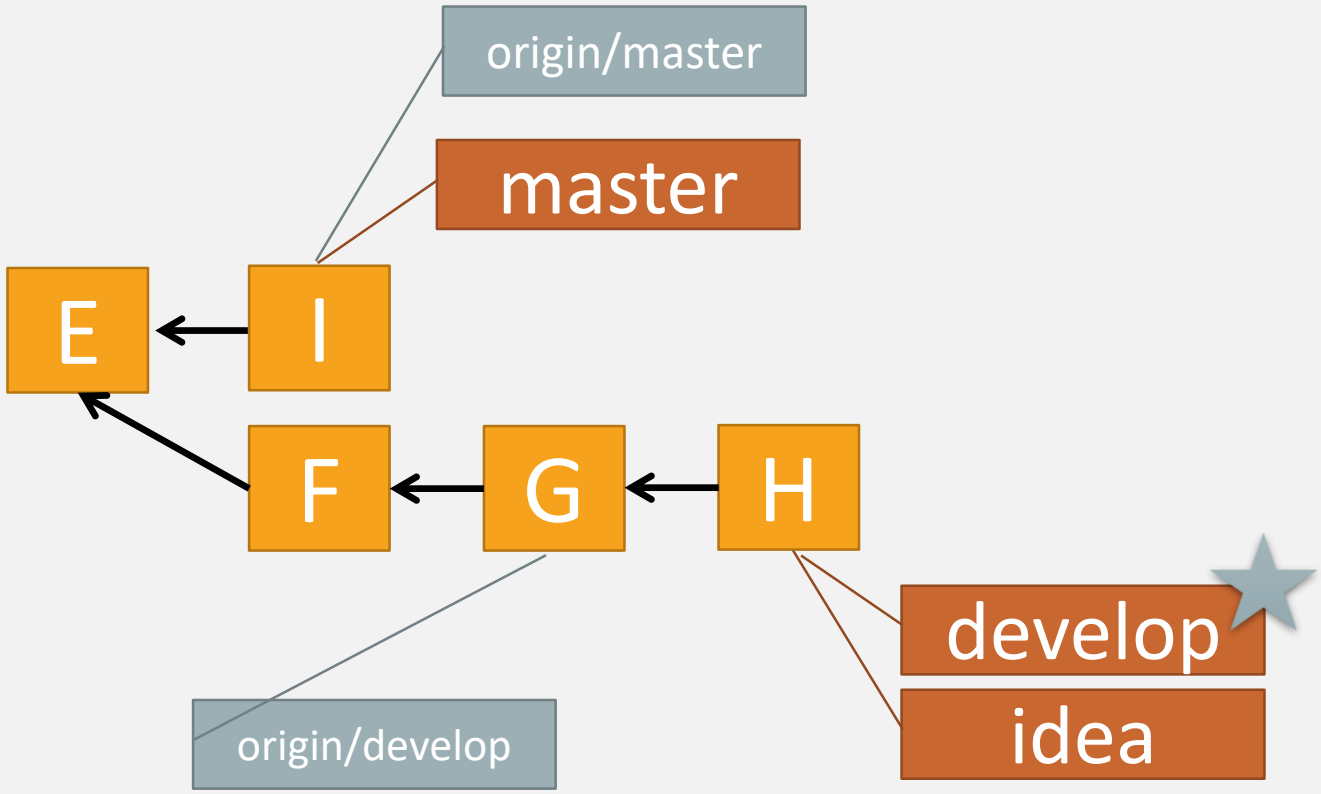

```
> git pull (at least daily)
```

BRANCHES ILLUSTRATED



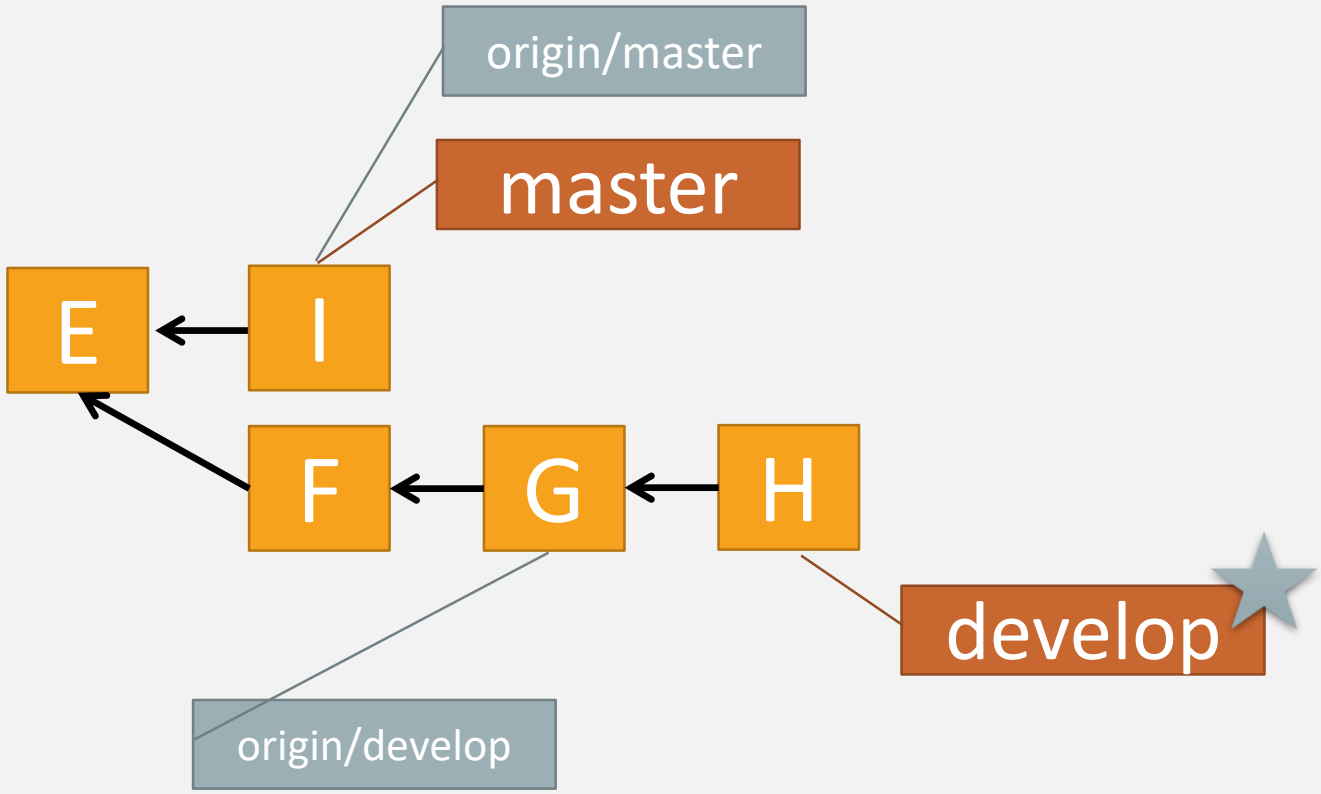

```
> git checkout develop
```

BRANCHES ILLUSTRATED



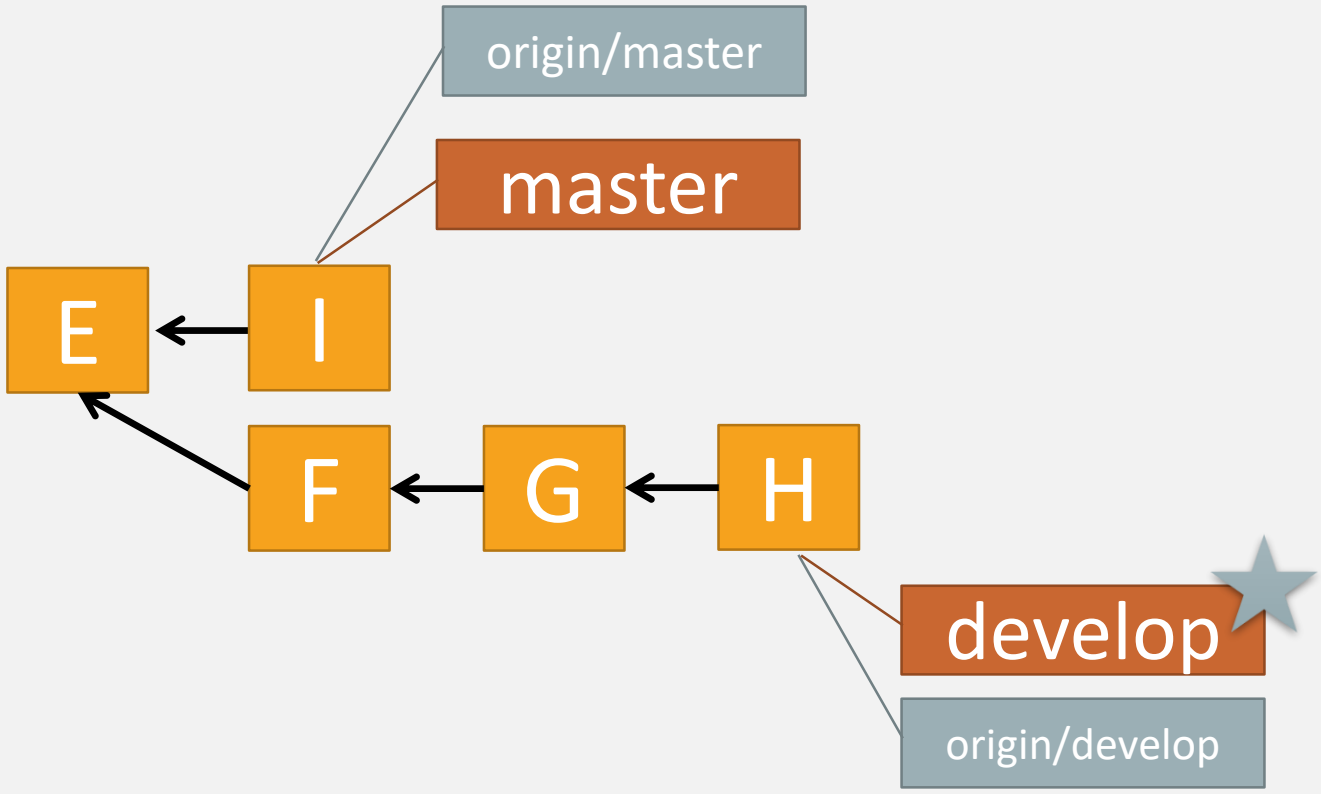

```
> git merge idea (fast forward merge)
```

BRANCHES ILLUSTRATED



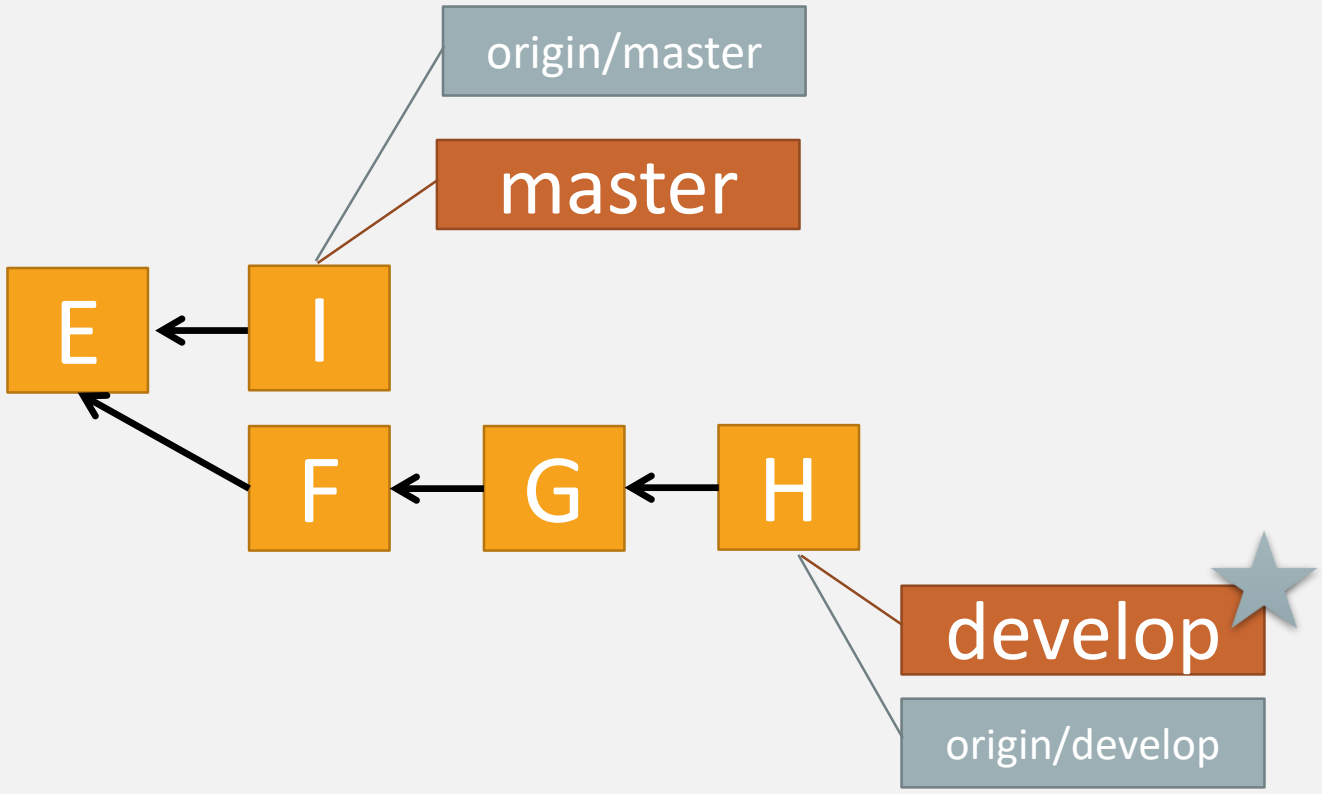

```
> git branch -d idea
```

BRANCHES ILLUSTRATED



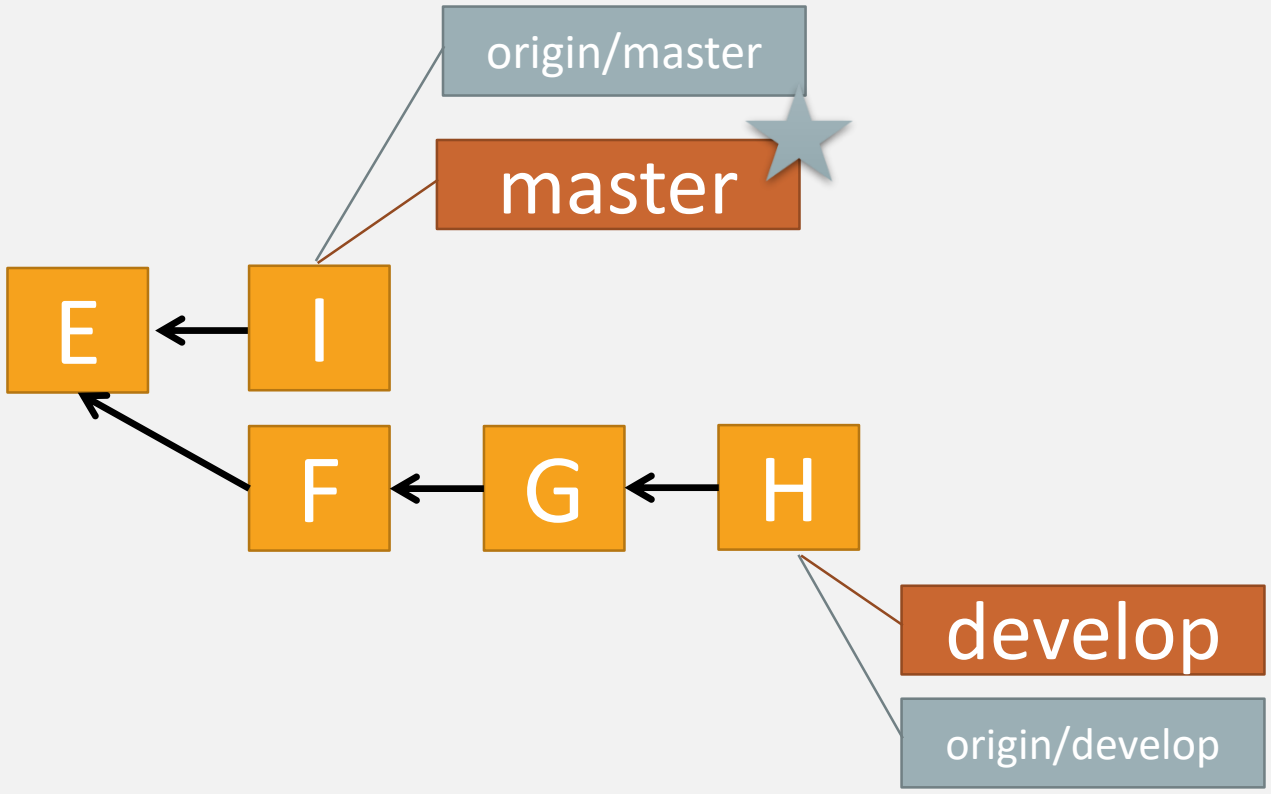

```
> git push origin develop
```

MERGE FLOW VS. REBASE FLOW



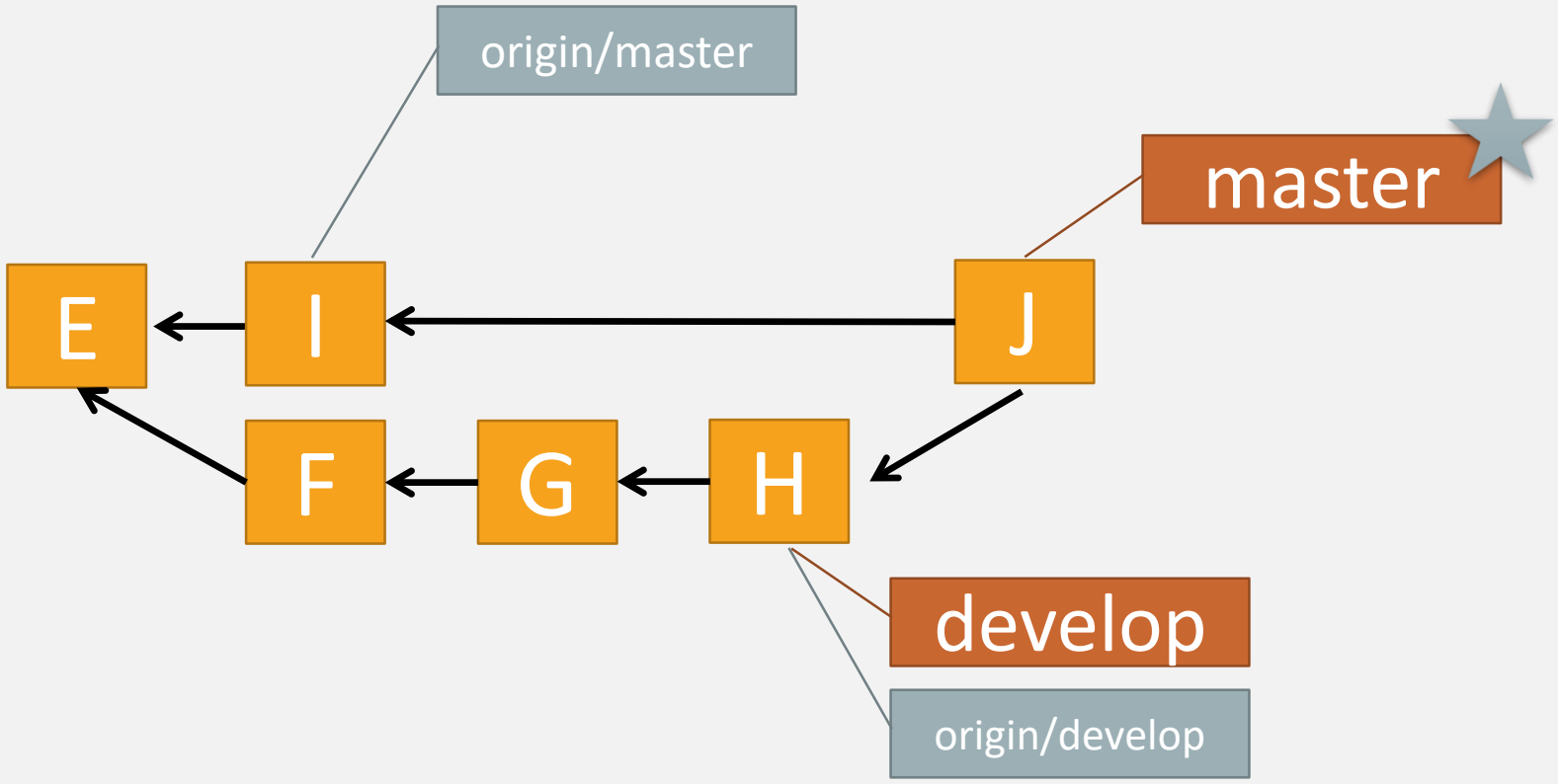

```
> git push origin develop
```

BRANCHES ILLUSTRATED – MERGE FLOW



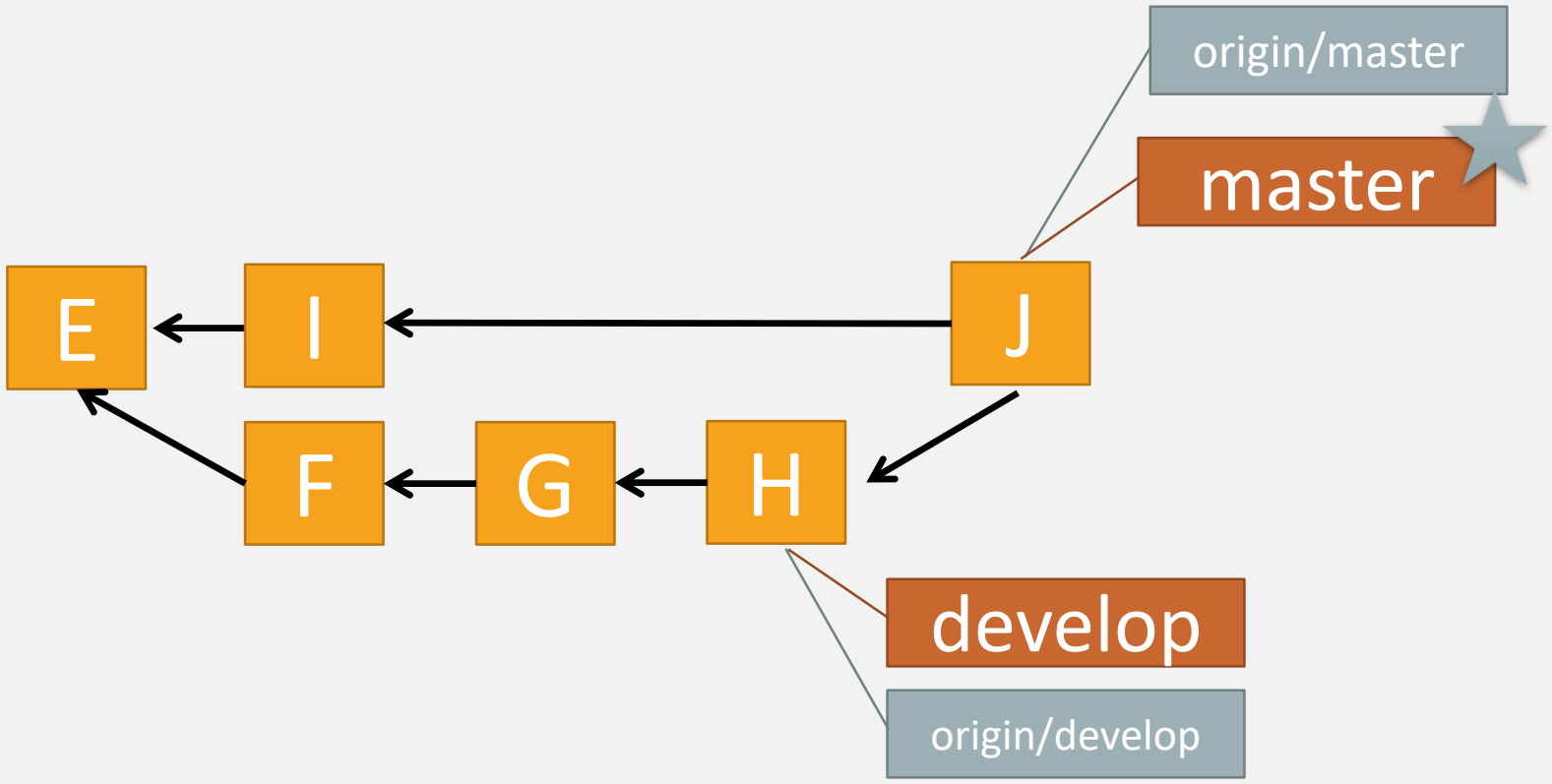

```
> git checkout master
```

BRANCHES ILLUSTRATED – MERGE FLOW



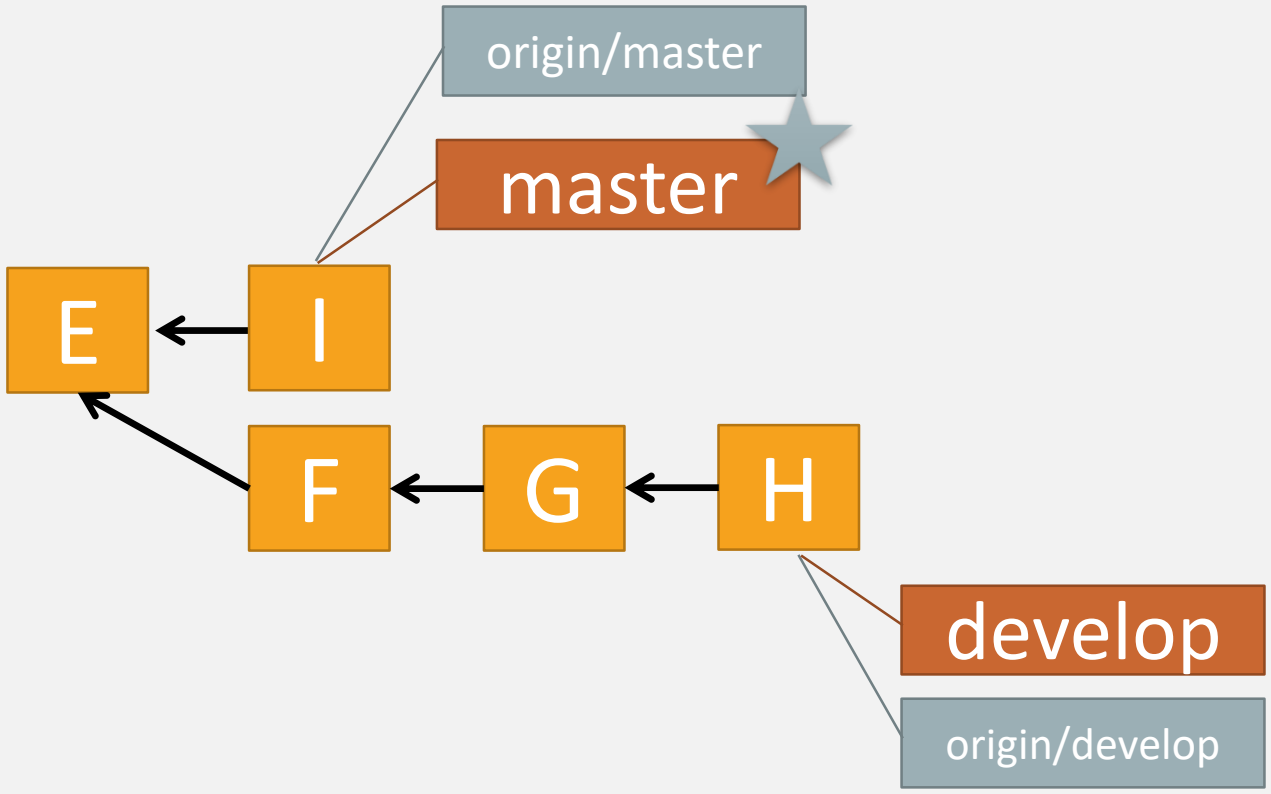

```
> git merge develop
```

BRANCHES ILLUSTRATED – MERGE FLOW



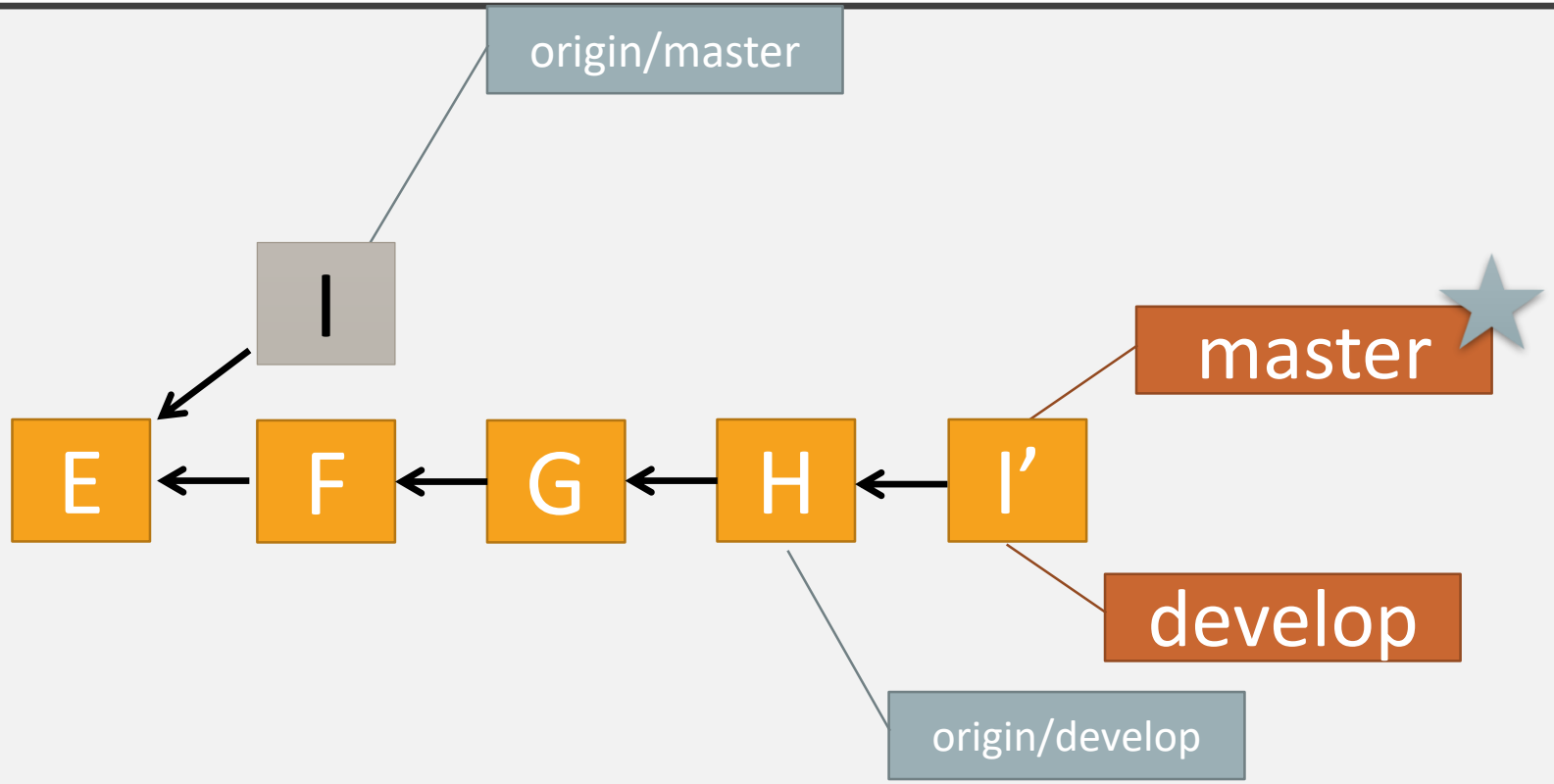

```
> git push origin
```

BRANCHES ILLUSTRATED – REBASE FLOW



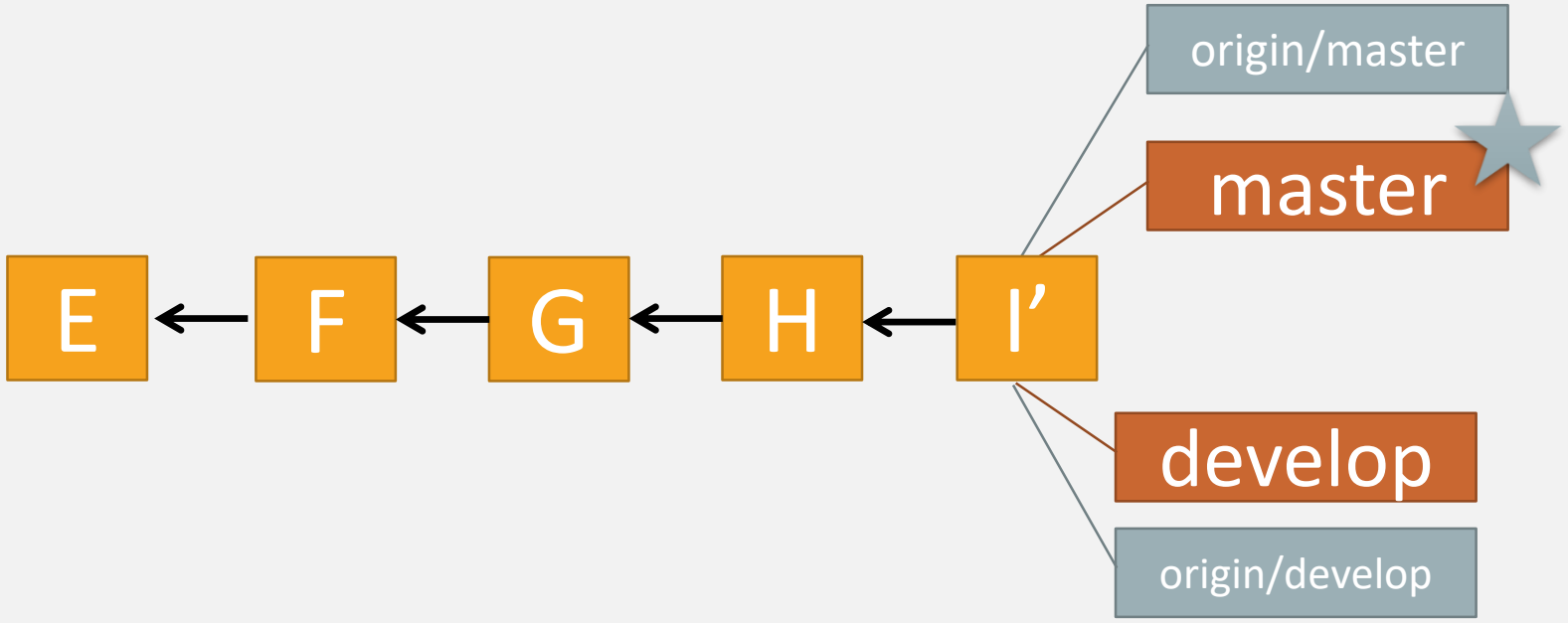

```
> git checkout master
```

BRANCHES ILLUSTRATED – REBASE FLOW



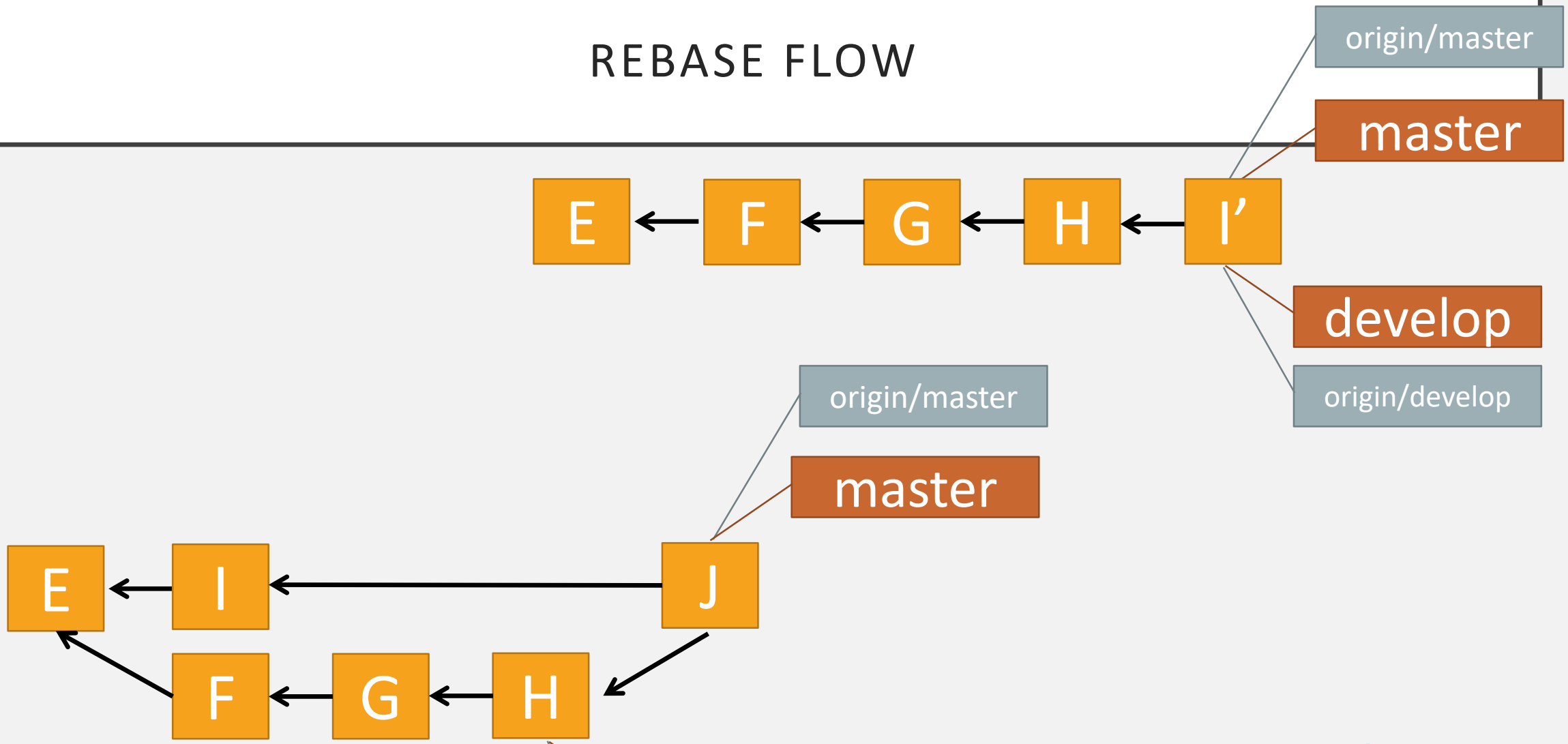

```
> git rebase develop
```

BRANCHES ILLUSTRATED – REBASE FLOW




```
> git push origin
```

REBASE FLOW



Merge Flow

SHORT VS. LONG-LIVED BRANCHES

- Great for multi-version work
- Follow same rules as Master
  - …use Story branches
- Define your conventions
  - What branches do you want to share?
  - Branch per environment?

DEPLOYING WITH GIT

DEPLOYING WITH GIT

Developer “FTP”

DEPLOYING WITH GIT

Developer “FTP”
Additional Branches pointing at:

DEPLOYING WITH GIT

Developer FTP

Additional Branches pointing at:

Test, Staging , Production

DEPLOYING WITH GIT

Developer FTP

Additional Branches pointing at:

Test, Staging , Production

Post Commit Hooks Automate deployments

CLOUD PROVIDERS – GIT SUPPORT

AppHarbor
Heroku
Nodejitsu
Windows Azure
… to name a few

GIT DEPLOYMENT

Simple workflow

GIT DEPLOYMENT

Simple workflow
Add Hooks to deploy on Commit

GIT DEPLOYMENT

Simple workflow

Add Hooks to deploy on Commit

Can get more advanced

GIT DEPLOYMENT

Simple workflow

Add Hooks to deploy on Commit

Can get more advanced

Add Build machines, push on success

http://Git-SCM.com

git --fast-version-control

Search entire site...

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.

Learn Git in your browser for free with Try Git.

About

The advantages of Git compared to other source control systems.

Documentation

Command reference pages, Pro Git book content, videos and other material.

Downloads

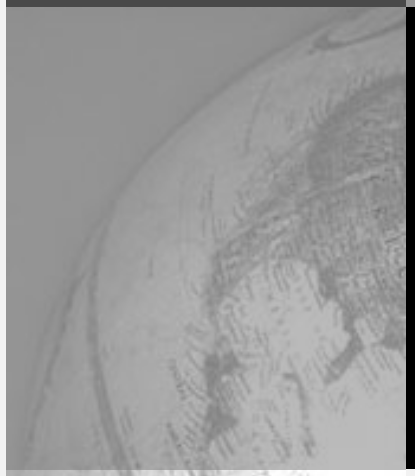
GUI clients and binary releases for all major platforms.

Community

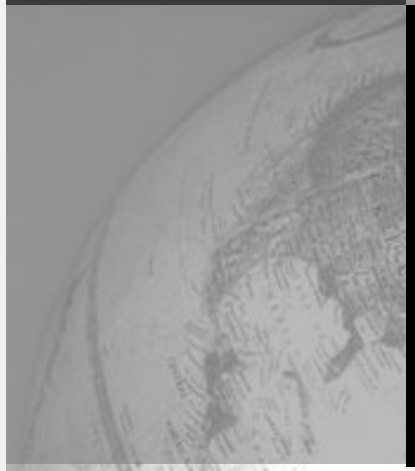
Get involved! Mailing list, chat, development and more.

TOOLS / RESOURCES

Pro Git (Book)	http://www.git-scm.com/book
TortoiseGit (with TortoiseMerge)	http://code.google.com/p/tortoisegit
Msysgit (includes git-bash)	http://code.google.com/p/msysgit
Posh-Git (for PowerShell users)	http://github.com/dahlbyk/posh-git
GitScc (Visual Studio integration)	http://gitscc.codeplex.com/
Windows Git Credential Store	http://gitcredentialstore.codeplex.com/
GitHub for Windows	http://windows.github.com/



Questions and Answers.



Thanks for listening!!!

Seshagiri Sriram
Email: SeshagiriSriram@gmail.com