An Introduction to GIT

Version Control with Git

Avoid to be the repository manager

Dr. Fabio Fumarola

Agenda

- What is Version Control? (and why use it?)
- What is Git? (And why Git?)
- How git works
- Create a repository
- Branches
- Add remote
- How data is stored

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

OR

Git is a

Directory

Git is a

Tree

history storage system

Git is a

DTK

Stupid

content tracker

How ever you think about it…

How ever you think about it…

GIT IS SUPER COOL

What is a Version Control

- Version Control - A system for managing changes made to documents and other computer files
- What kinds of files can we use it with?
  - Source code
  - Documentation
  - Short stories
  - Binary files (music and pictures)
- What should we use it for?
  - Text files
  - Projects that have lots of revisions (changes)
  - Collaborating

VCS Systems and their Operations

- Lots of different choices available:
  - CVS
  - SVN
  - Perforce
  - Git
  - Mercurial (Hg), Bazaar
  - And more!
- Most follow a repository model (though differ in how the repositories work)

So why do we need a VCS?

- Our goals
  - Share code (or something else) easily
  - Keep track of any changes we make (and undo them with ease)
  - Maintain multiple versions of the same project/code base
  - Clearly communicate what changes have been made
- There are two type of VCS
  - Centralized and
  - Distributed

Distributed vs Centralized

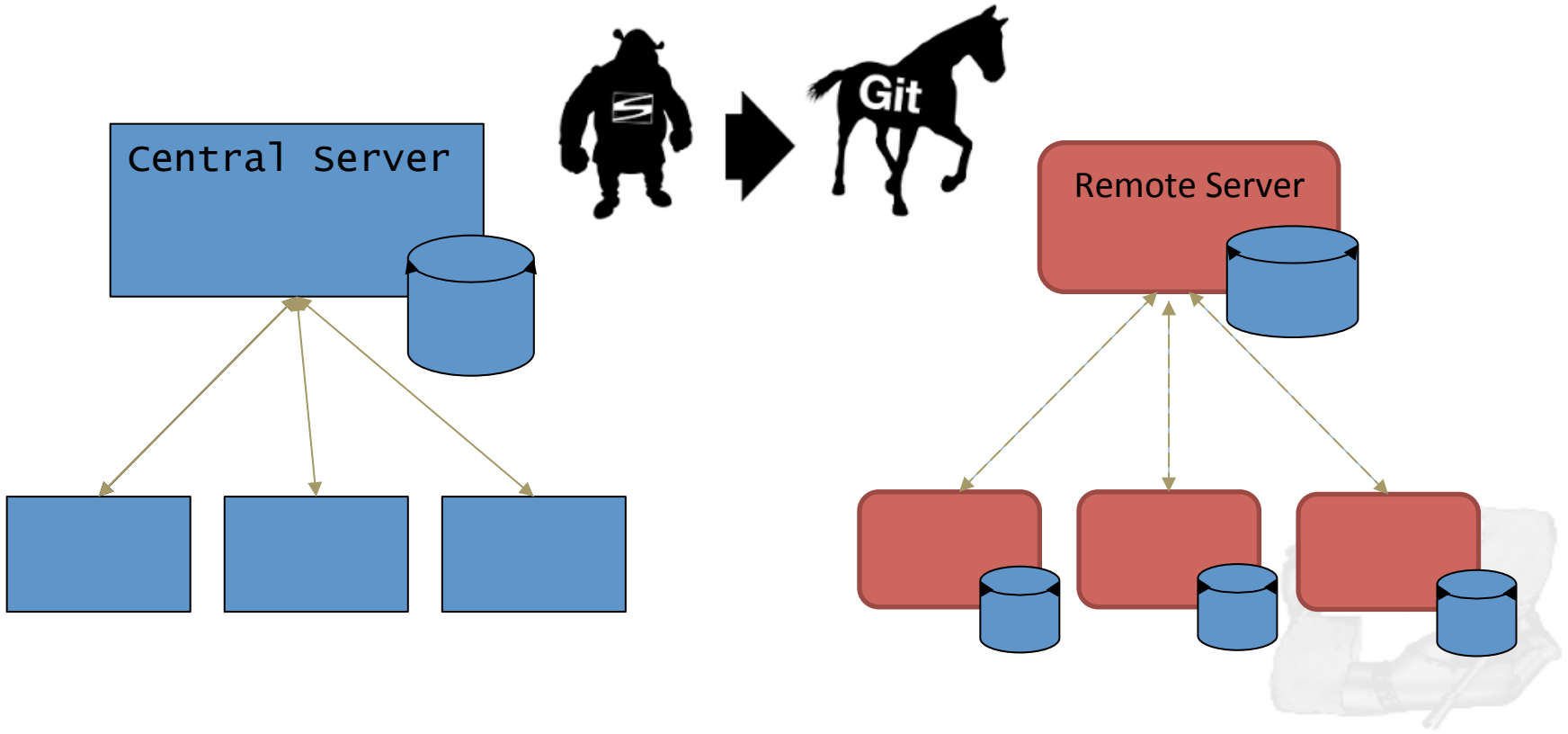
Centralized VCS

- One central repository
- Must be capable of connecting to repo
- Need to solve issues with group members making different changes on the same files

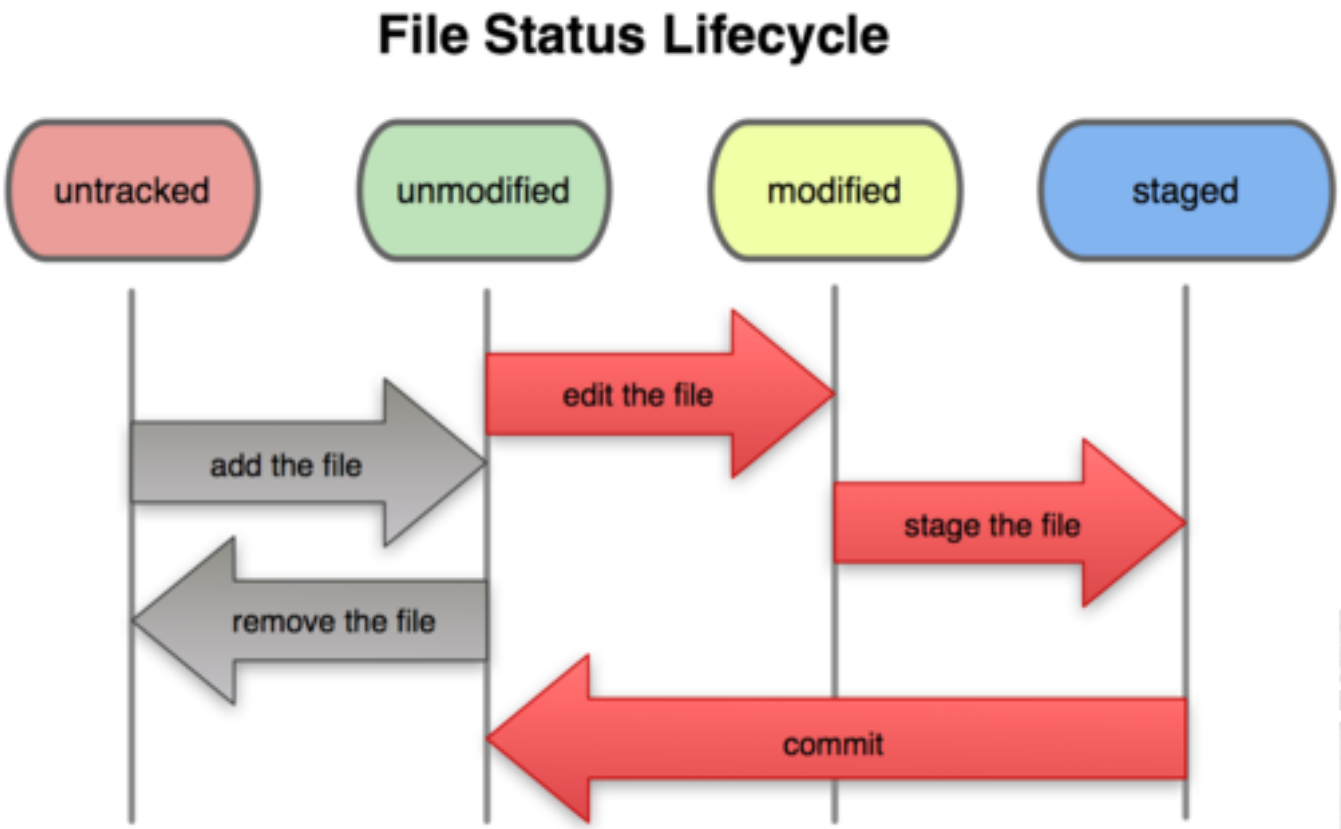
Distributed VCS

- Everyone has a working repo
- Faster
- Connectionless
- Still need to resolve issues, but it's not an argument against DVCS

Centralized vs Distributed



Git file lifecycle



Creating our first repository

- Install git
- Establish our first repository:
  - mkdir git-test
  - cd git-test
  - git init
- What do we have here?
  - git status

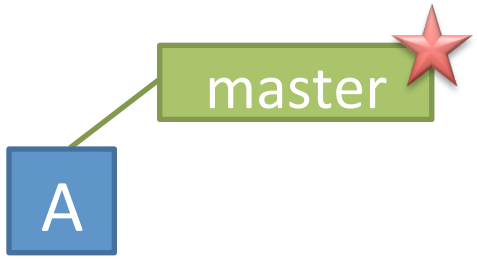
Using our first repository

- Add a file
  - touch file.txt
  - git add file.txt
  - git commit –m “add the first file”

Branching

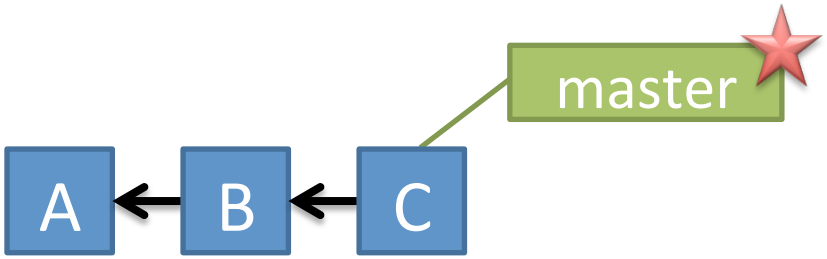
- With Git we should embrace the idea of branching
- Branching is the best method to work with other in a project

Branches Illustrated



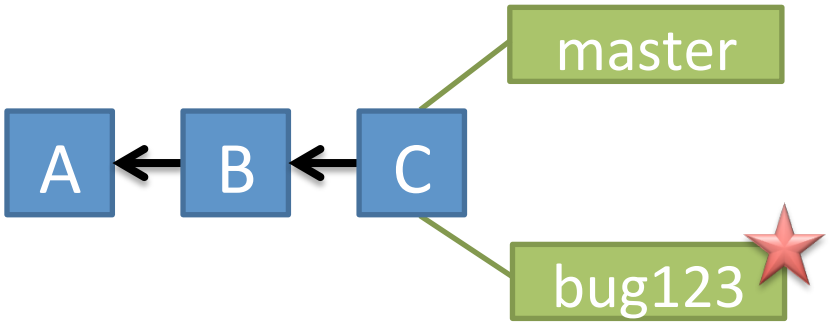

```
> git commit –m ‘my first commit’
```

Branches Illustrated



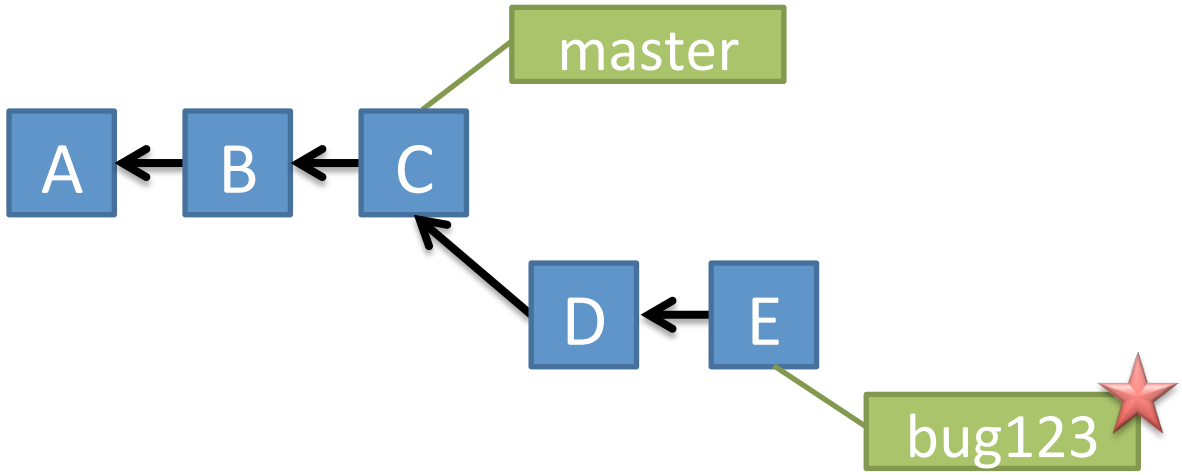

```
> git commit (x2)
```

Branches Illustrated



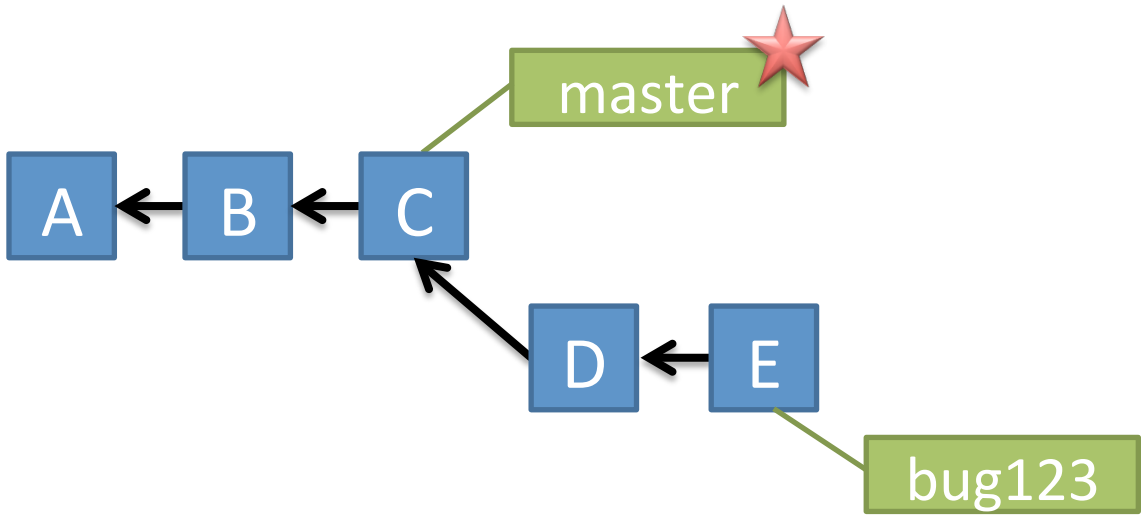

```
> git checkout -b bug123
```

Branches Illustrated



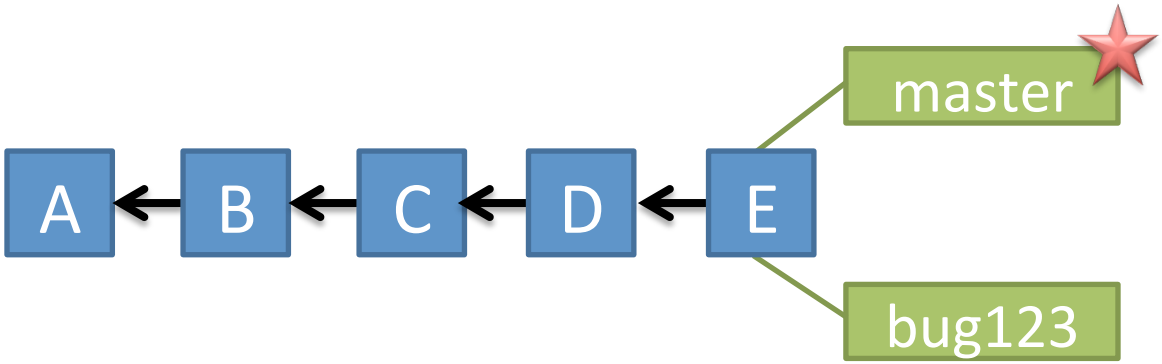

```
> git commit (x2)
```

Branches Illustrated



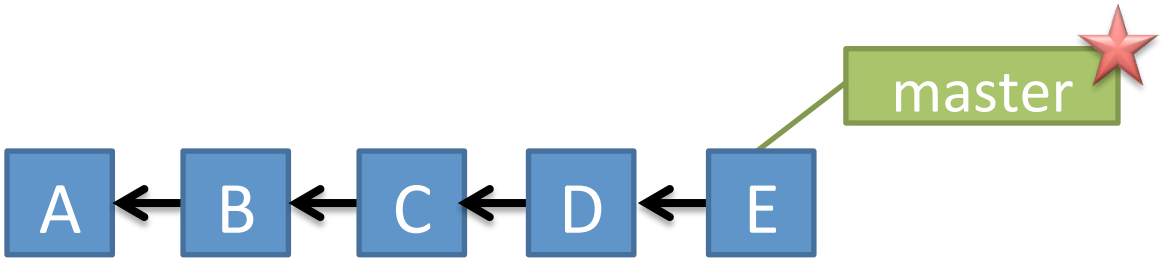

```
> git checkout master
```

Branches Illustrated



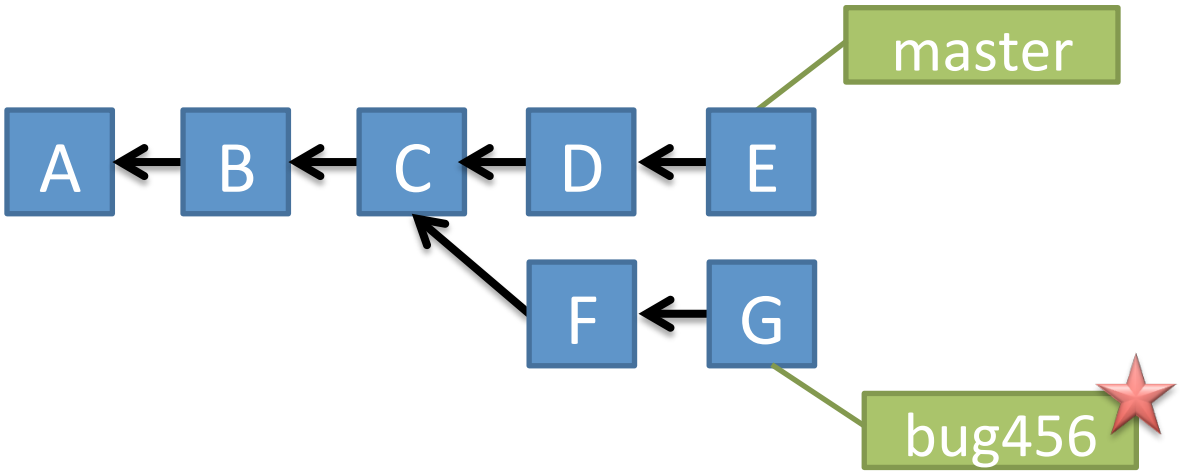

```
> git merge bug123
```

Branches Illustrated

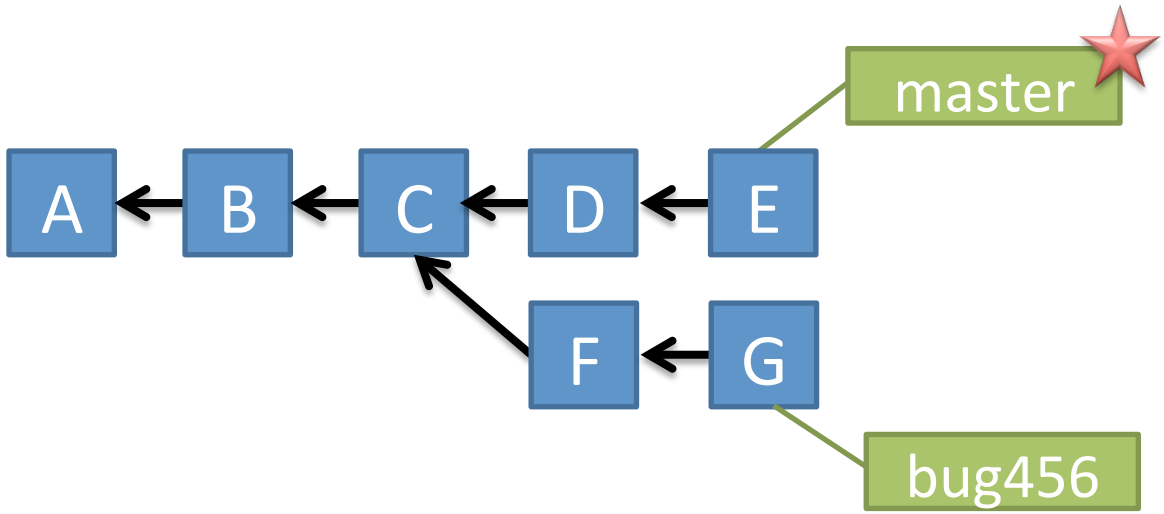



```
> git branch -d bug123
```

Branches Illustrated

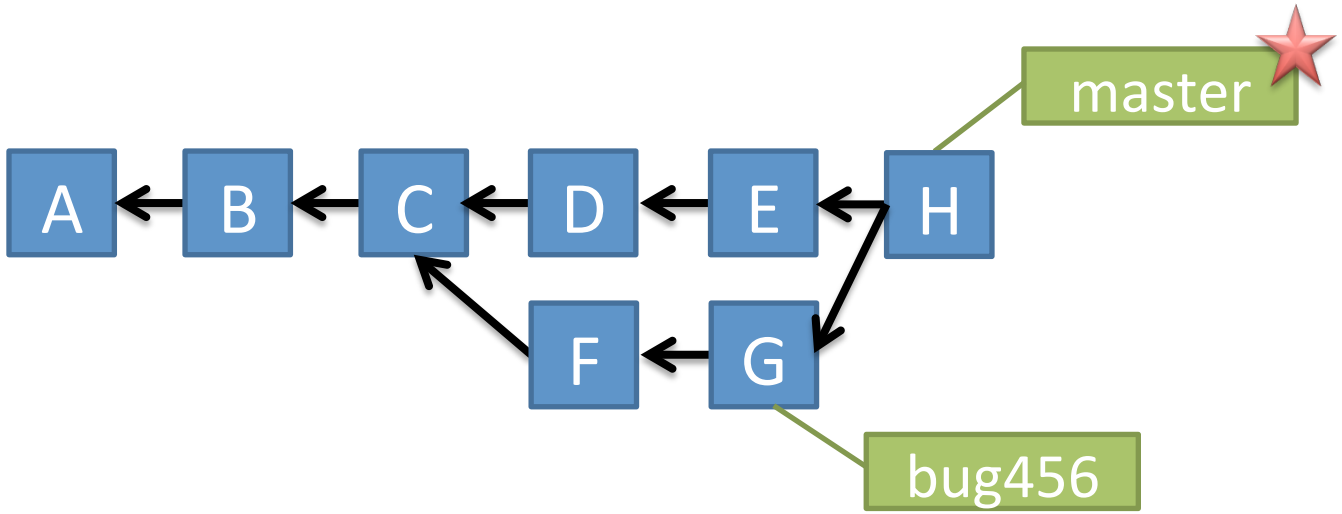


Branches Illustrated



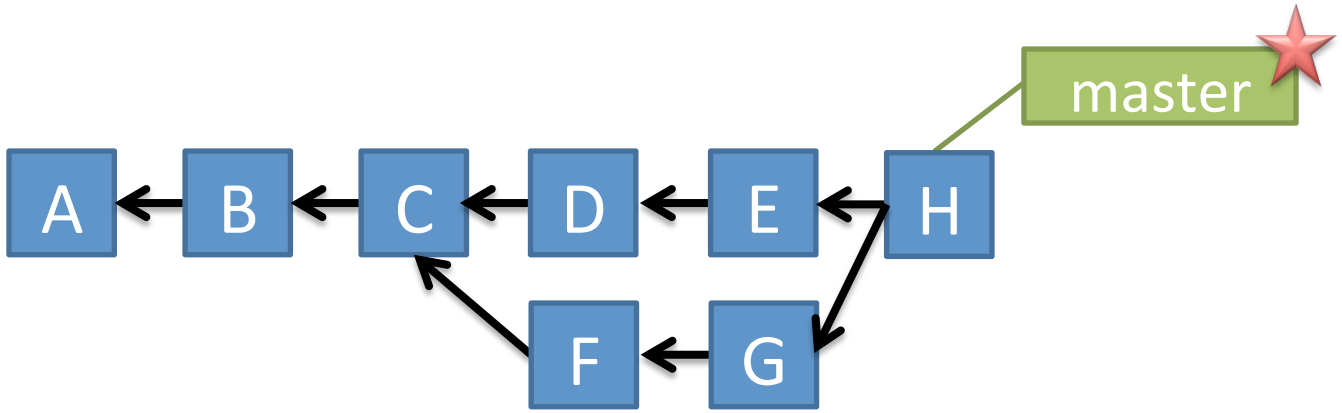

```
> git checkout master
```

Branches Illustrated



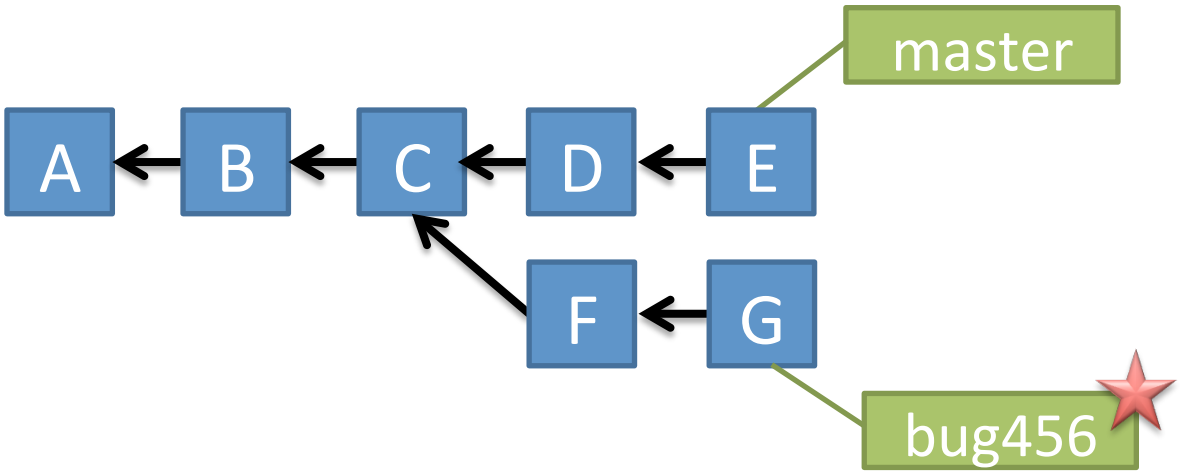

```
> git merge bug456
```

Branches Illustrated

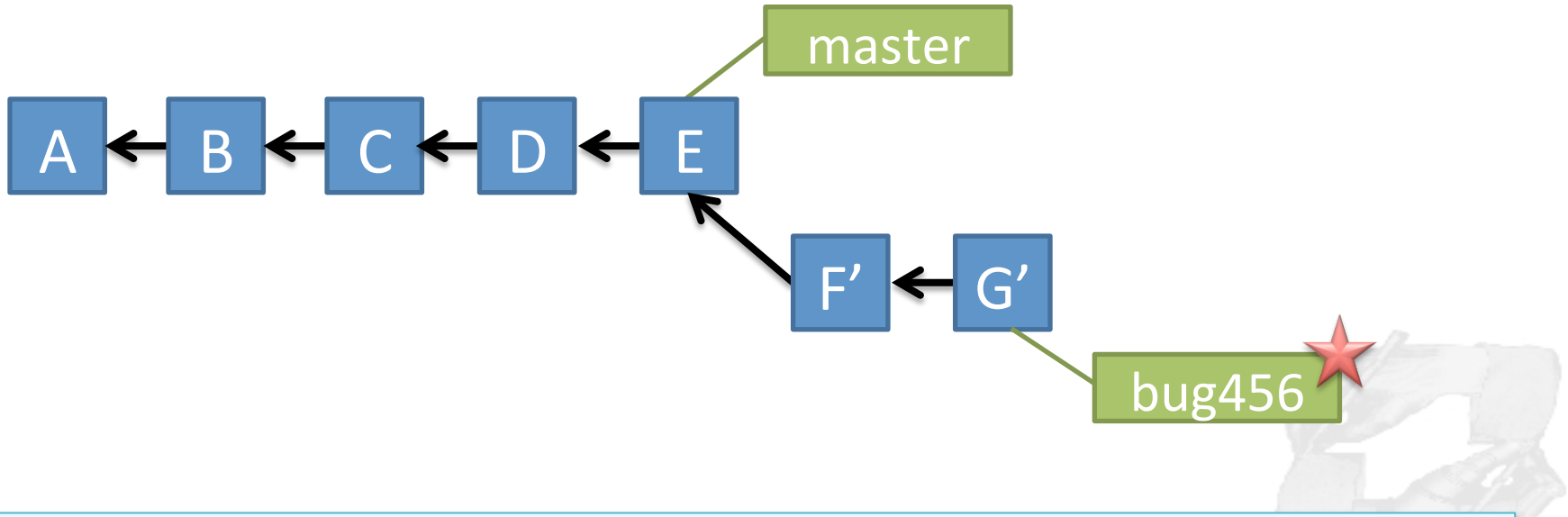



```
> git branch -d bug456
```

Branches Illustrated

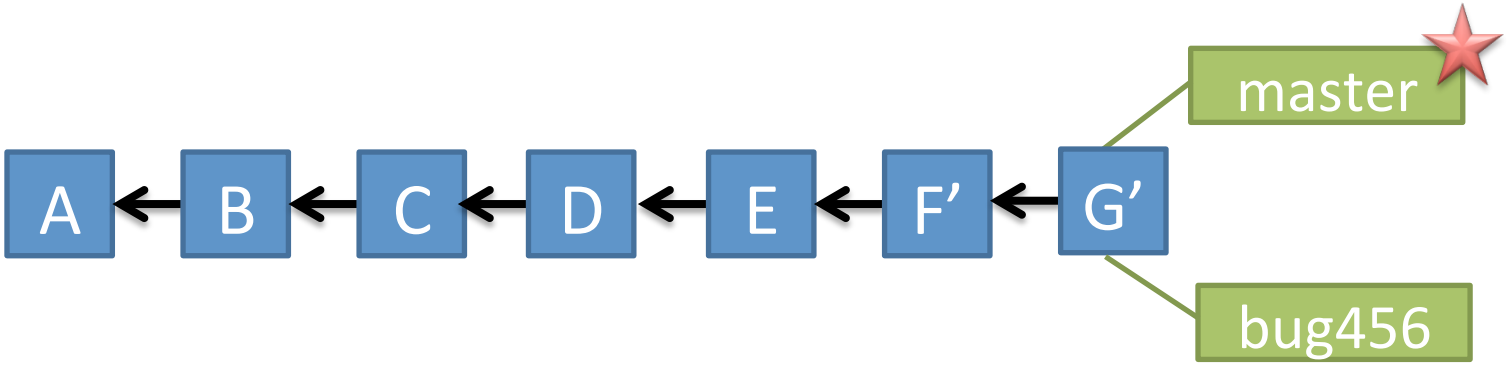


Branches Illustrated




```
> git rebase master
```

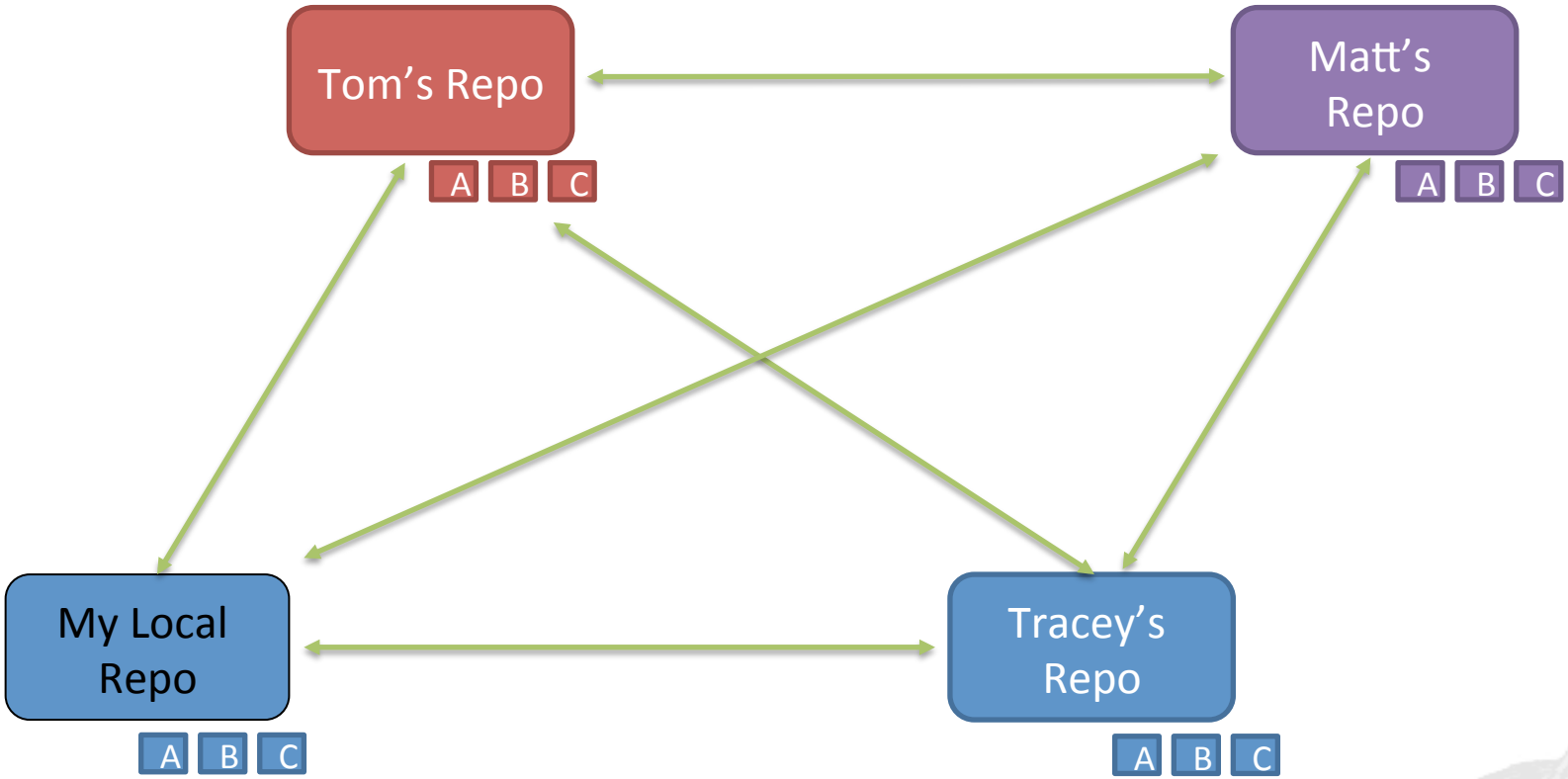
Branches Illustrated




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

Branches Review

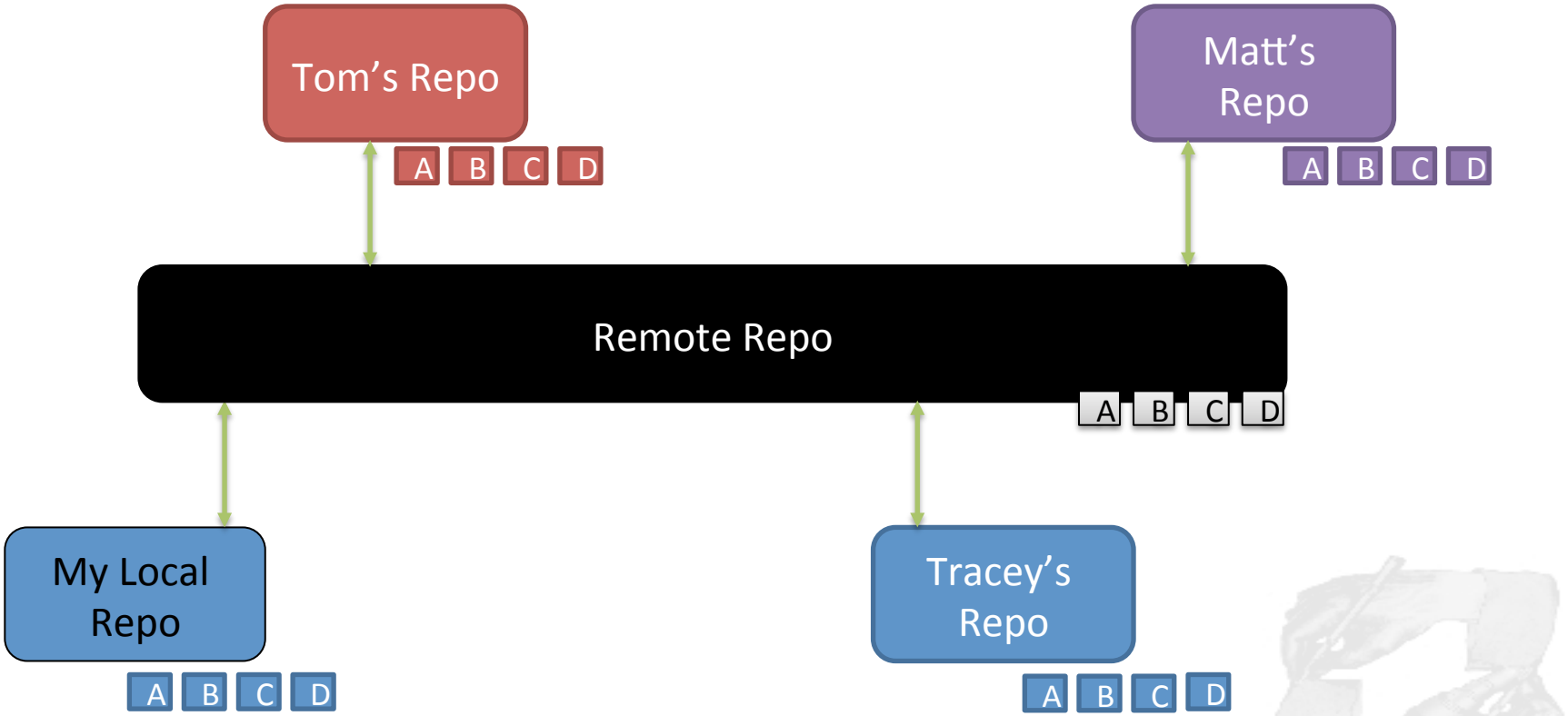
- Branches should be used to create “Features” in our project
- Local branches are very powerful
- Rebase is not scary



ADDING A REMOTE

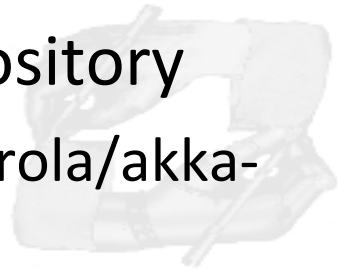
But, how we share our code with the other collaborators?

Sharing commits

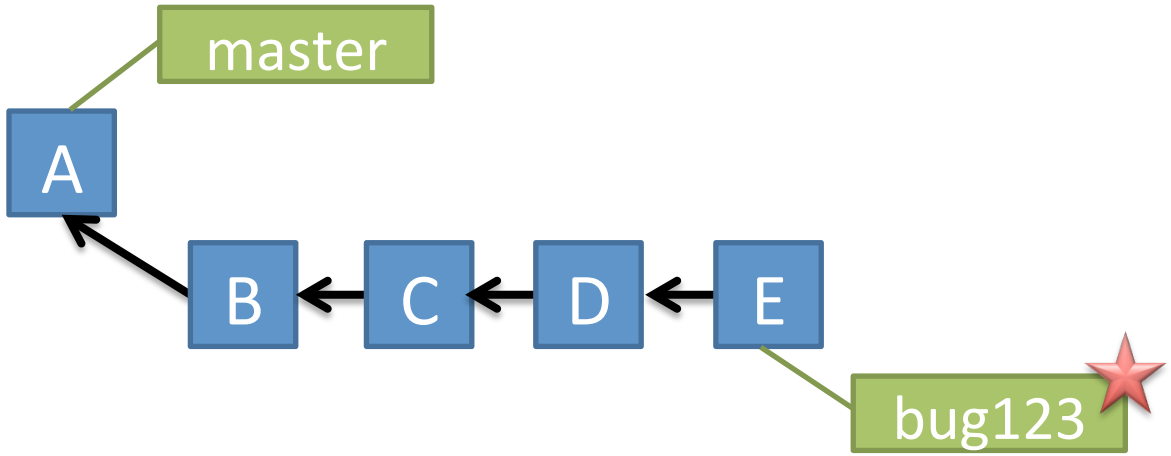


Setting up a Remote

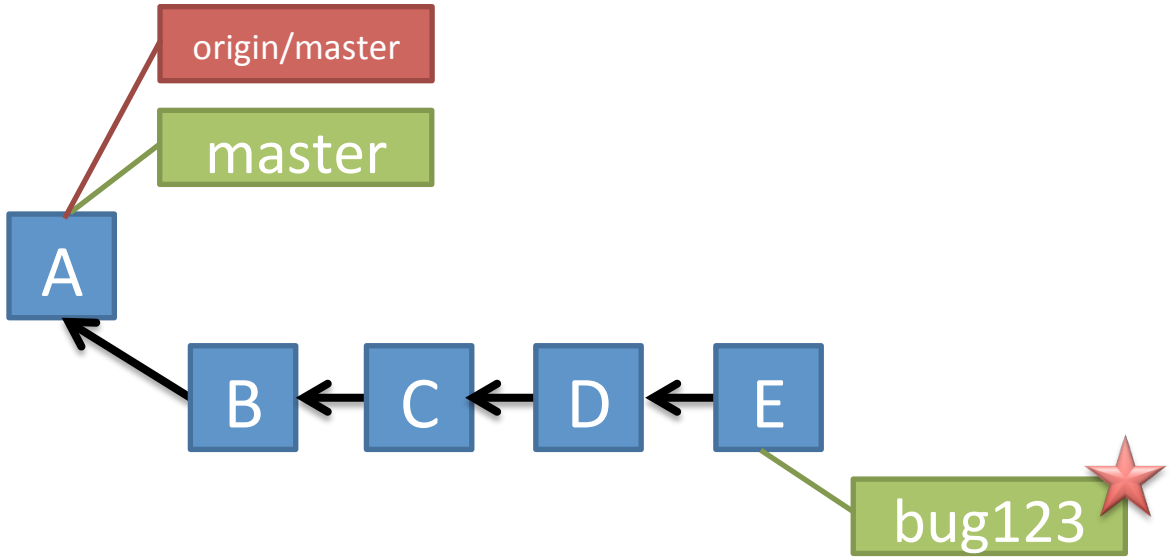
- We can clone an existing repository
  - git clone git@github.com:fabiofumarola/akka-tutorial.git
- We can push our changes
  - git push
- We can pull friends change
  - git pull
- We can also add a remote to an existing repository
  - git remote add origin git@github.com:fabiofumarola/akka-tutorial.git



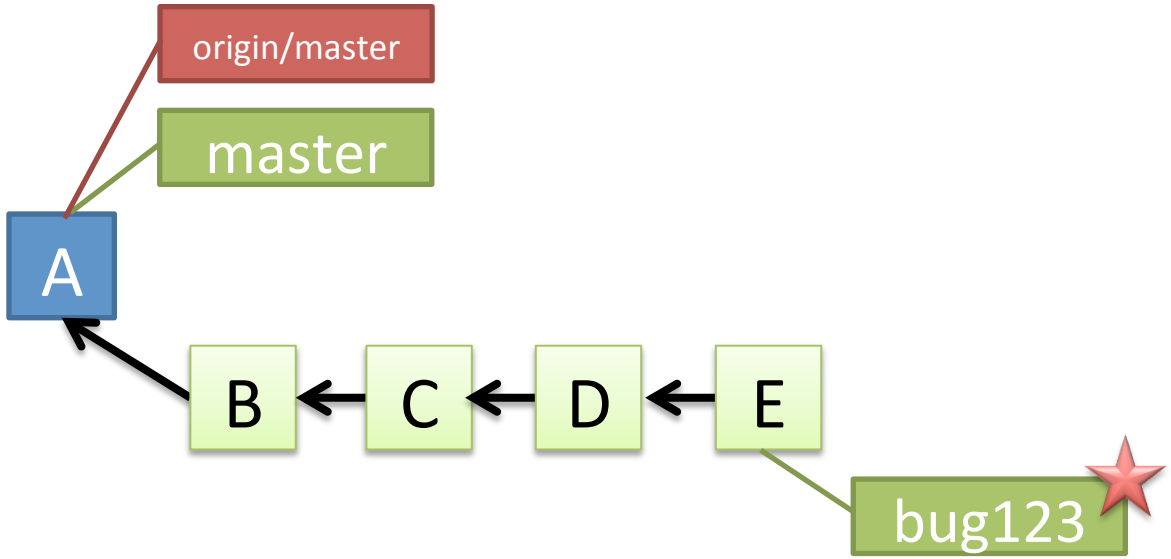
Branches with remote



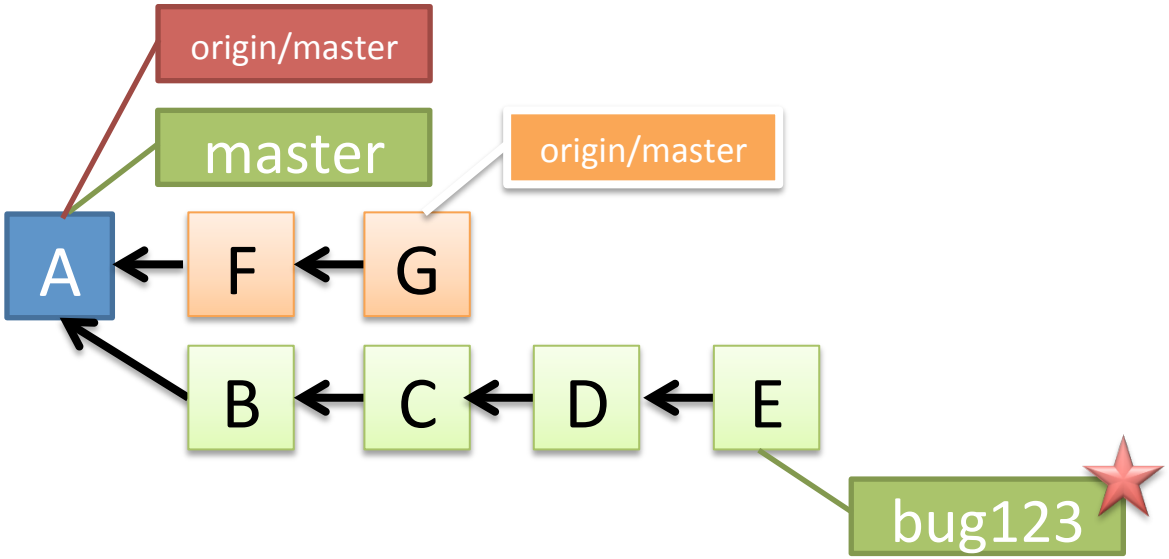
Branches with remote



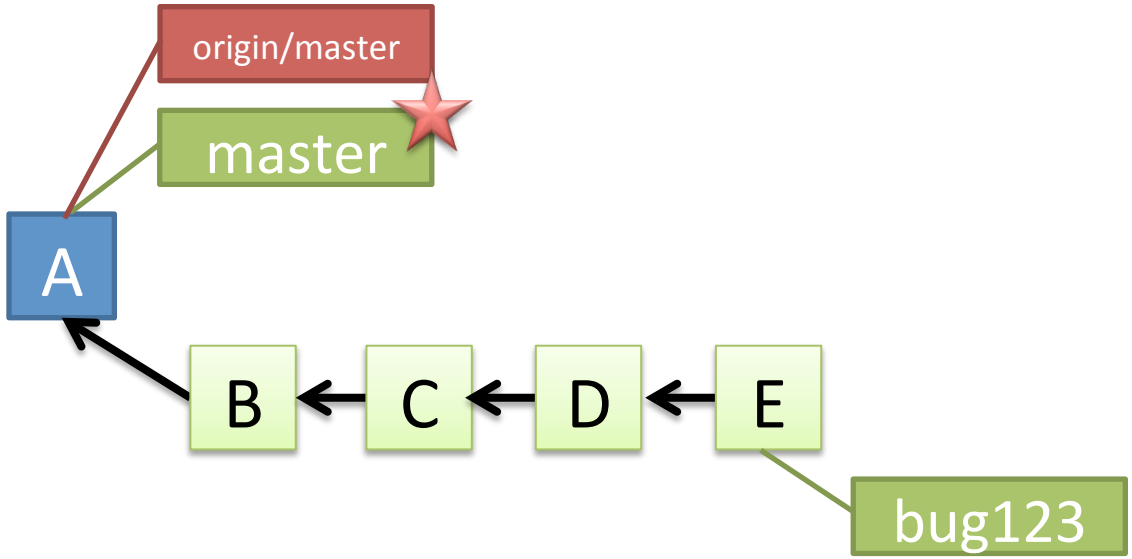
Branches with remote



Branches with remote

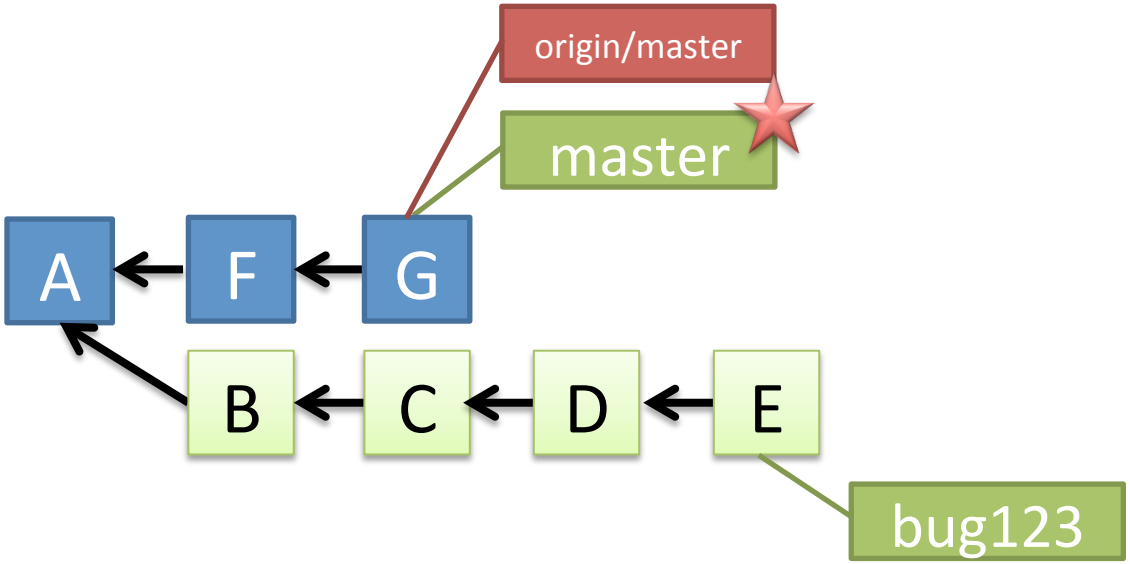


Branches with remote




```
> git checkout master
```

Branches with remote



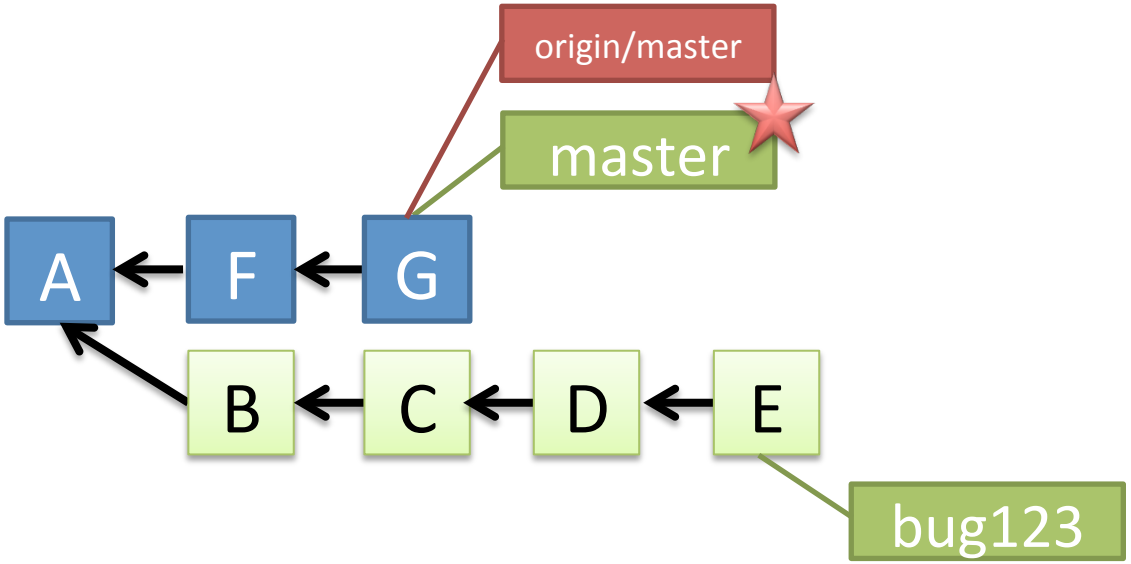

```
> git pull origin
```

Pull = Fecth + Merge

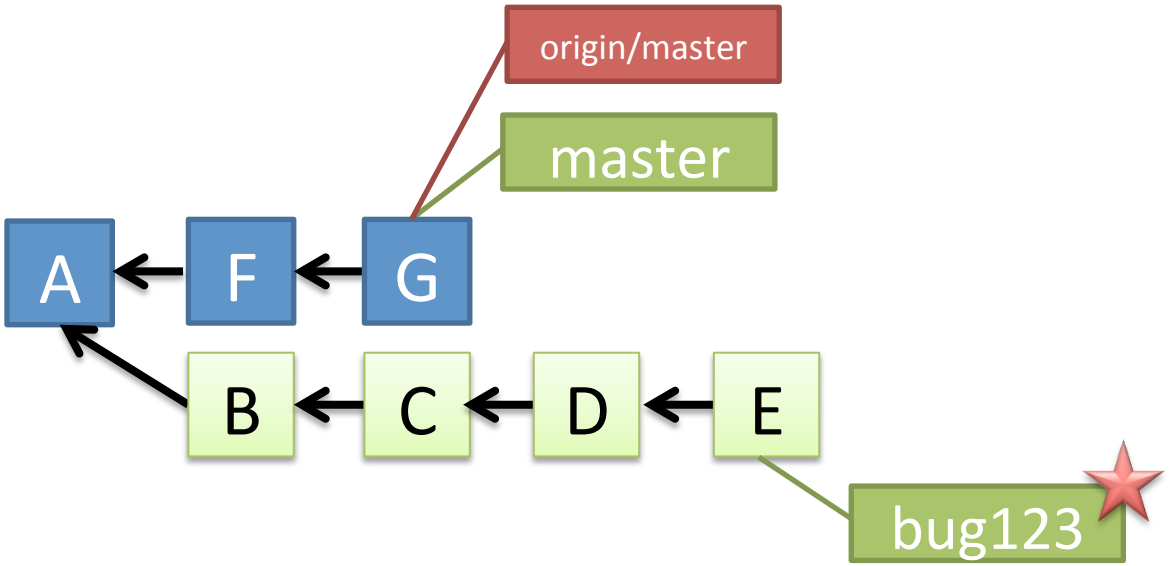
Fetch - updates your local copy of the remote branch

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

Branches Illustrated

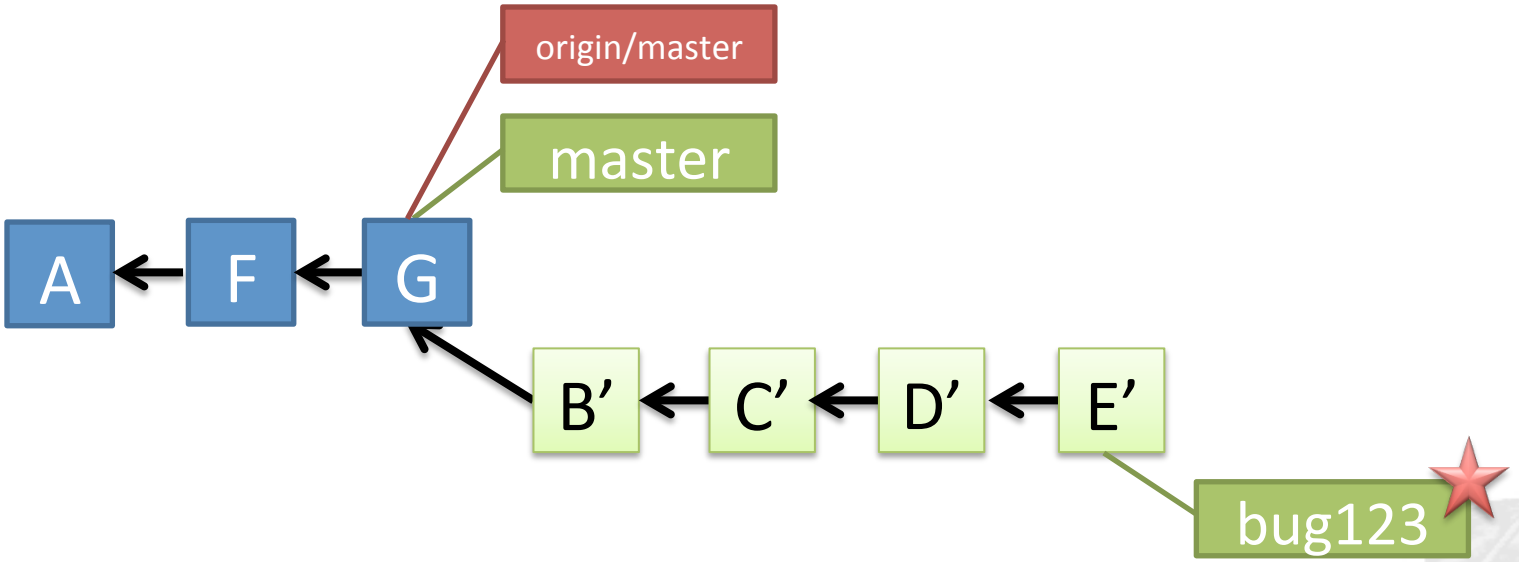


Branches Illustrated



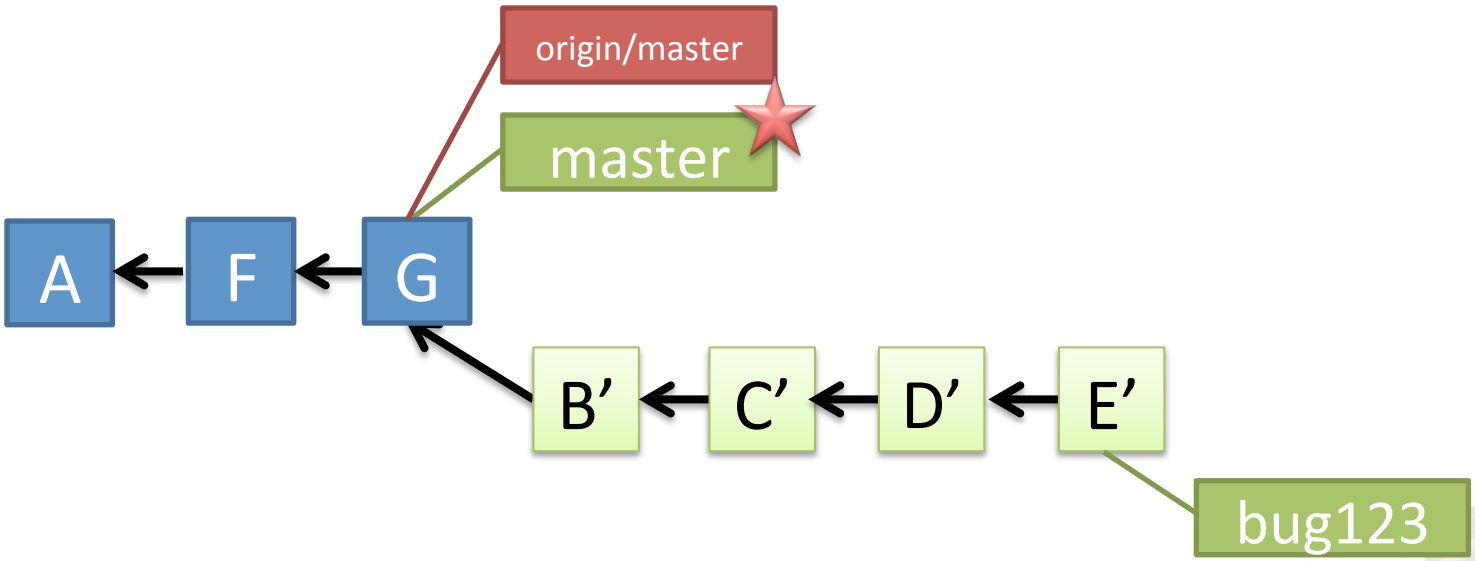

```
> git checkout bug123
```

Branches Illustrated



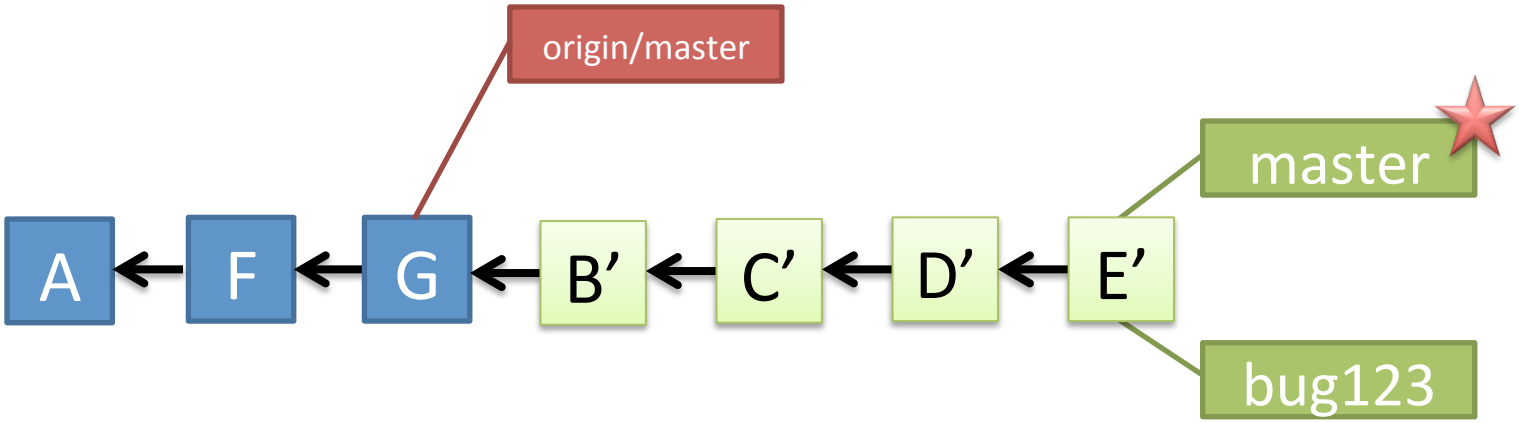

```
> git rebase master
```

Branches Illustrated



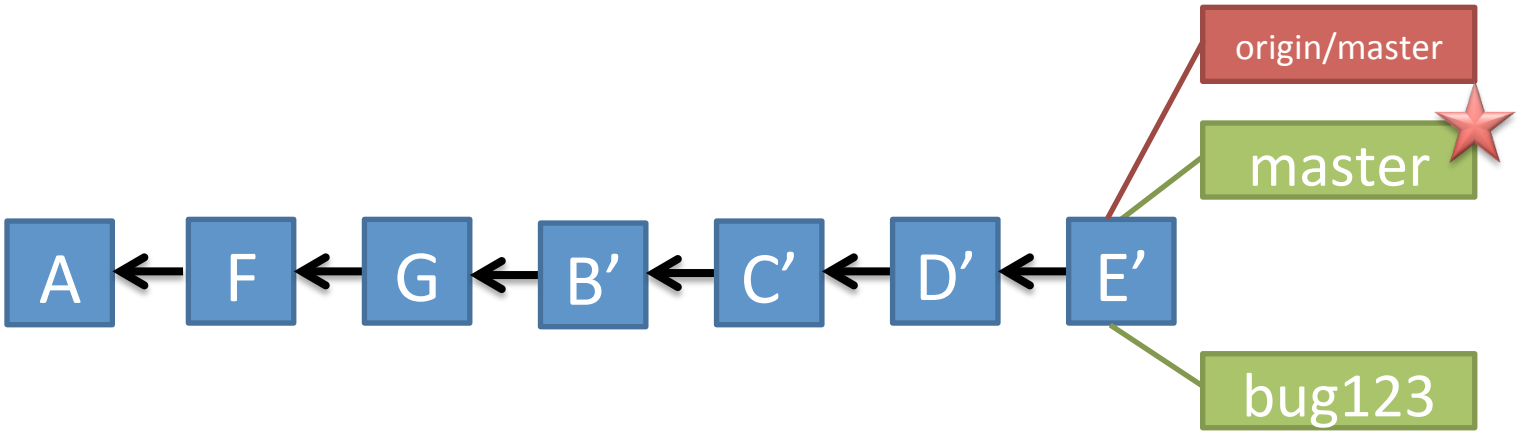

```
> git checkout master
```

Branches Illustrated




```
> git merge bug123
```

Branches Illustrated




```
> git push origin
```

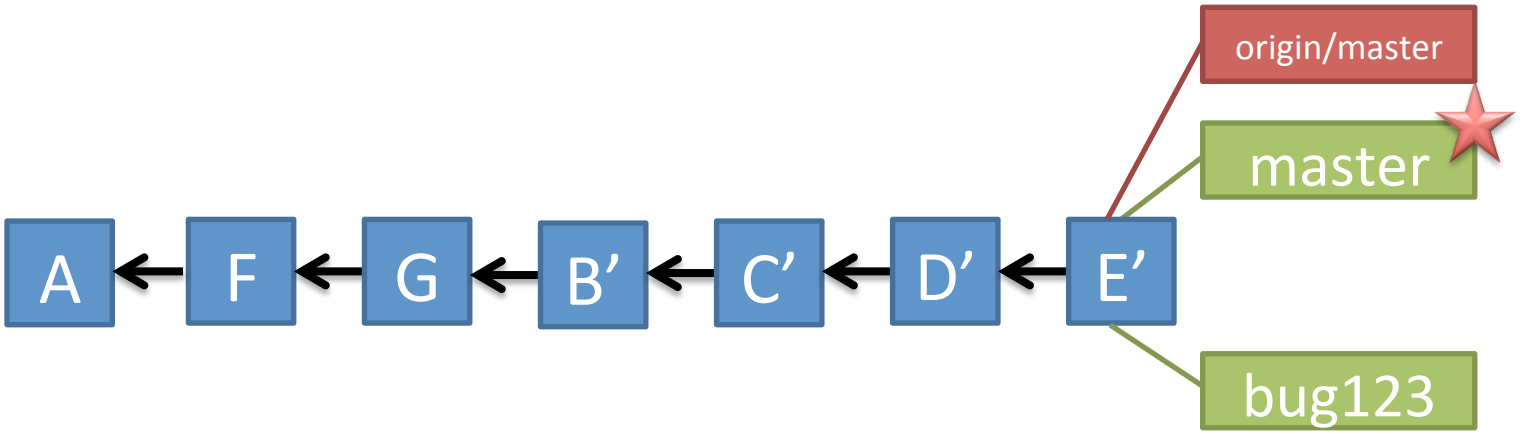
Push

Pushes your changes upstream

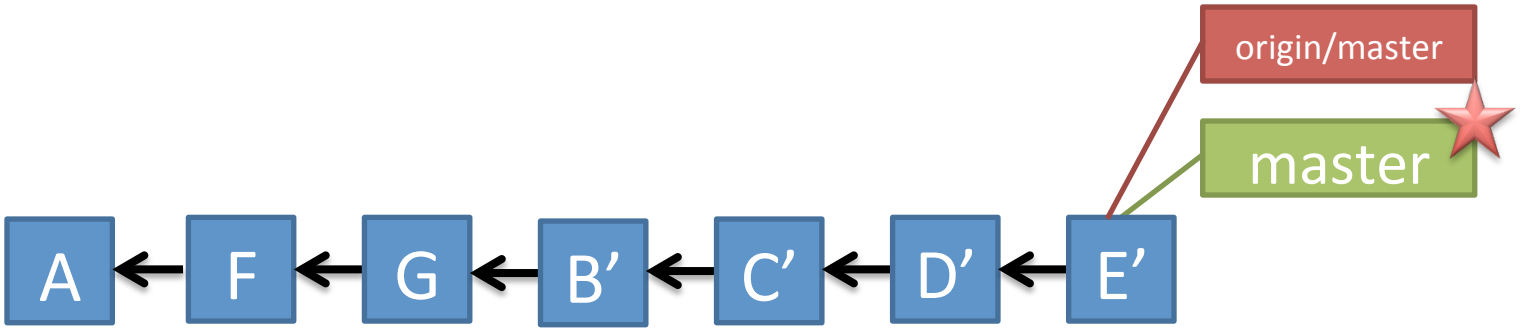
Git will reject pushes if newer changes exist on remote.

Good practice: Pull then Push

Branches Illustrated



Branches Illustrated

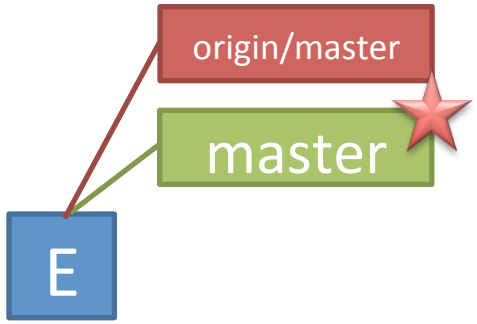



```
> git branch -d bug123
```

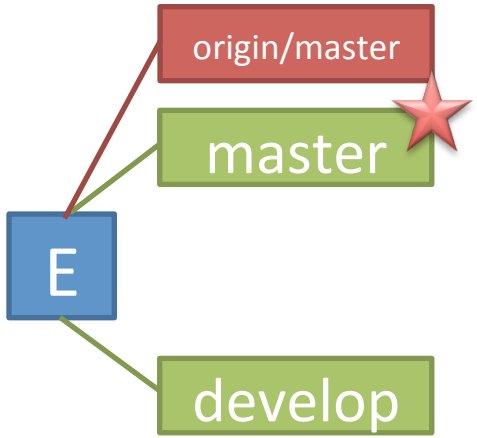
Short vs. Long-Lived Branches

- We can use branch to:
  - Solve bugs (hotfixes)
  - Create features
  - Make a release
- In order to simplify the management we can use:
  - Git Flow: http://danielkummer.github.io/git-flow-cheatsheet/index.it_IT.html

Branches Illustrated

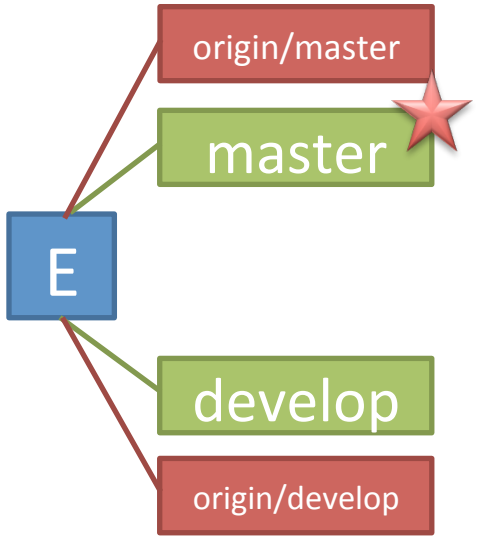


Branches Illustrated



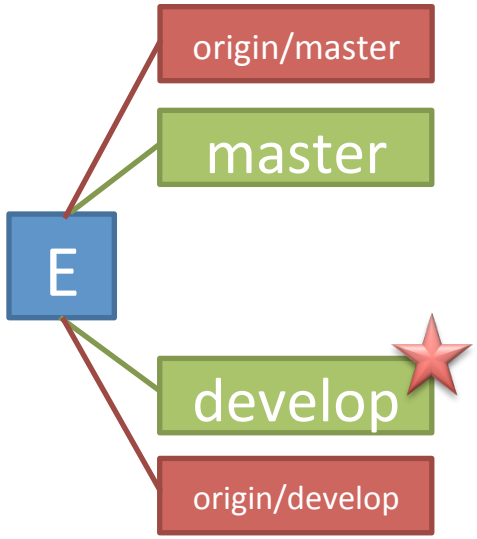

```
> git branch develop
```

Branches Illustrated



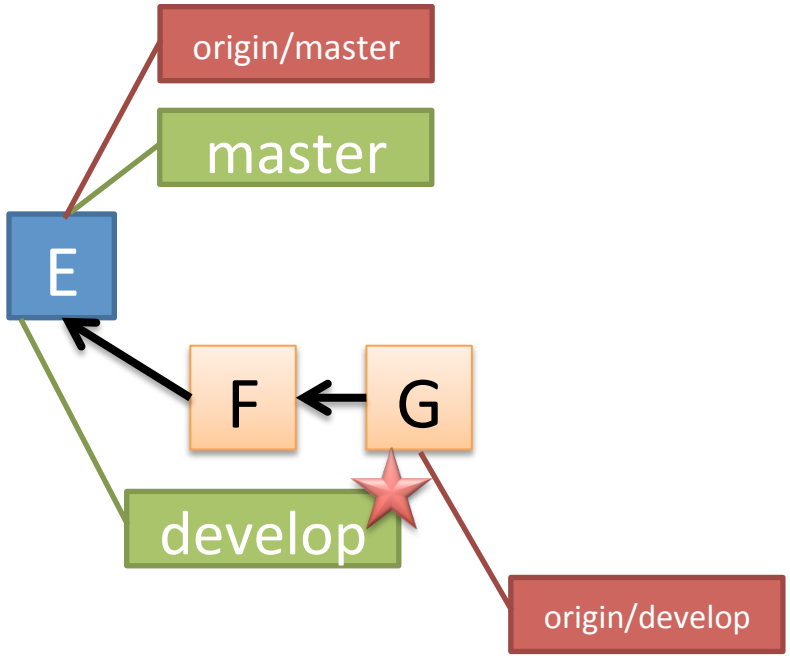

```
> git push origin develop
```

Branches Illustrated

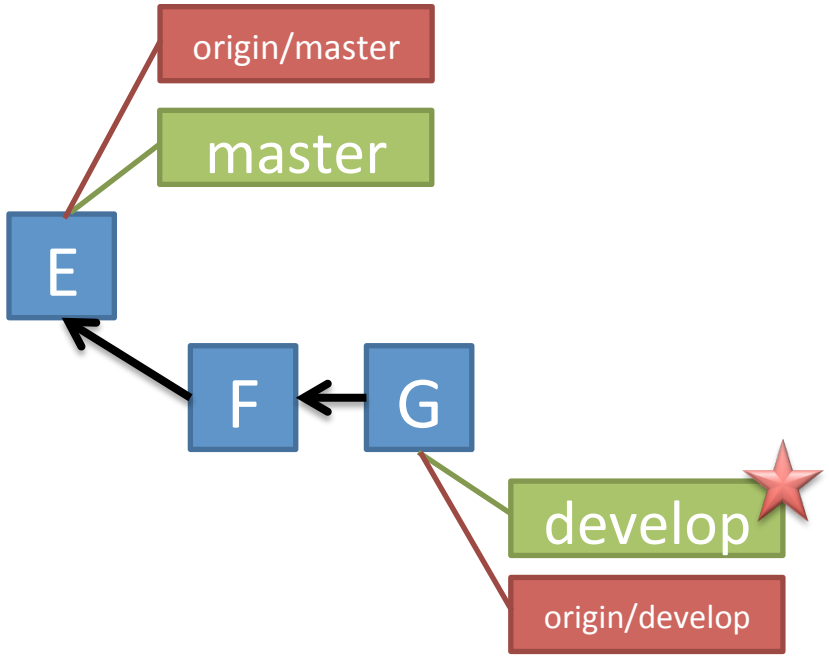



```
> git checkout develop
```

Branches Illustrated

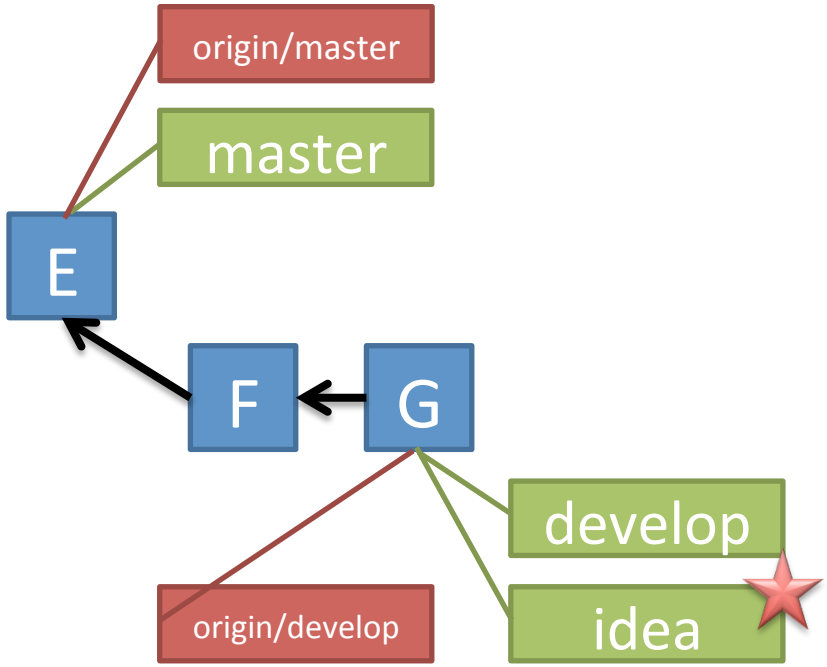


Branches Illustrated



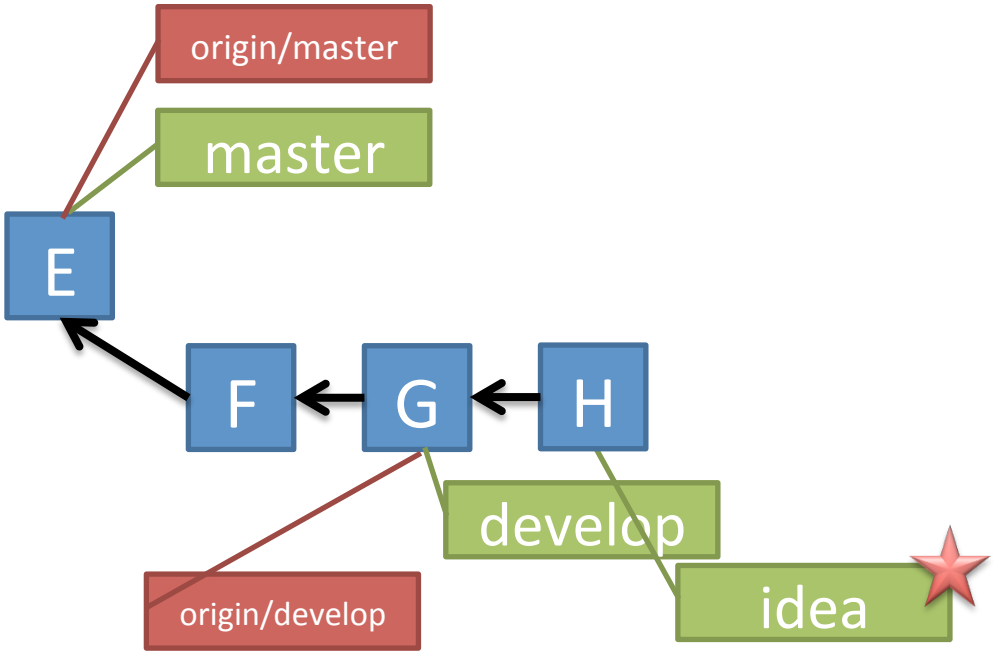

```
> git pull origin develop
```

Branches Illustrated



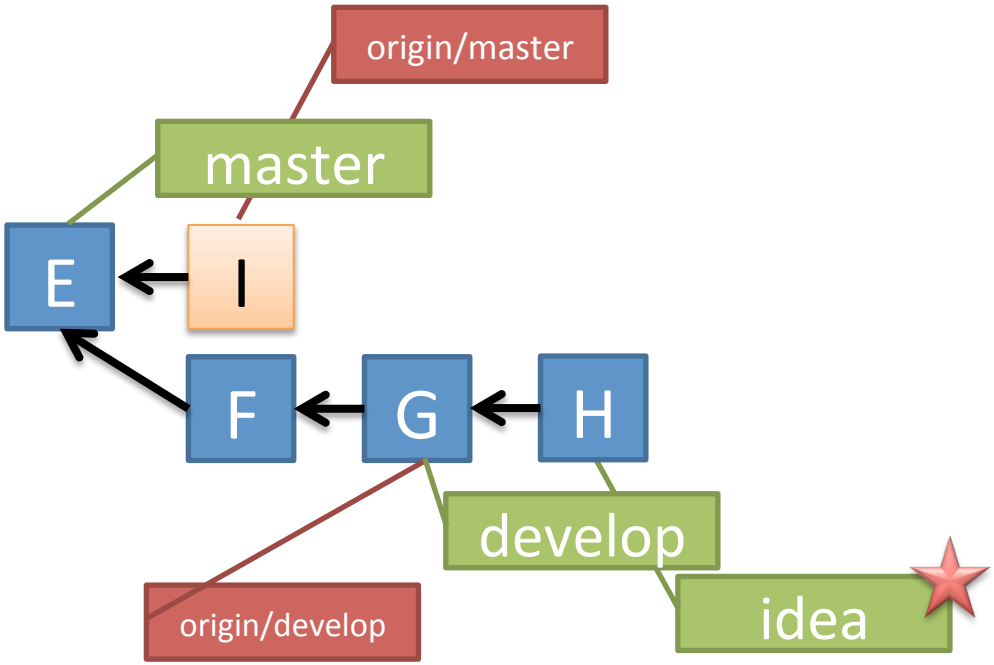

```
> git checkout -b idea
```

Branches Illustrated

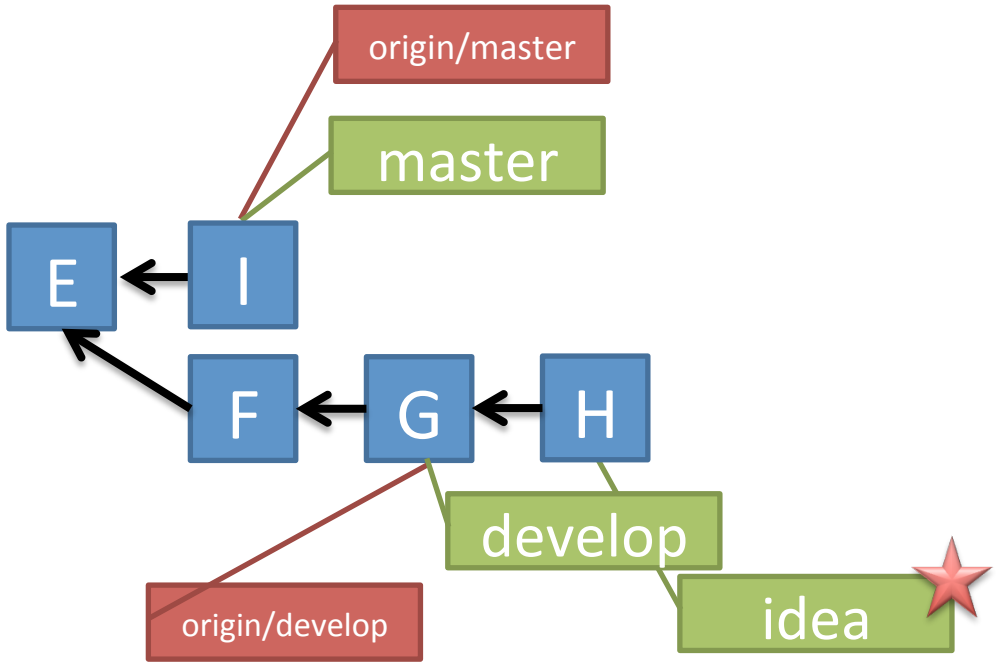



```
> git commit
```

Branches Illustrated

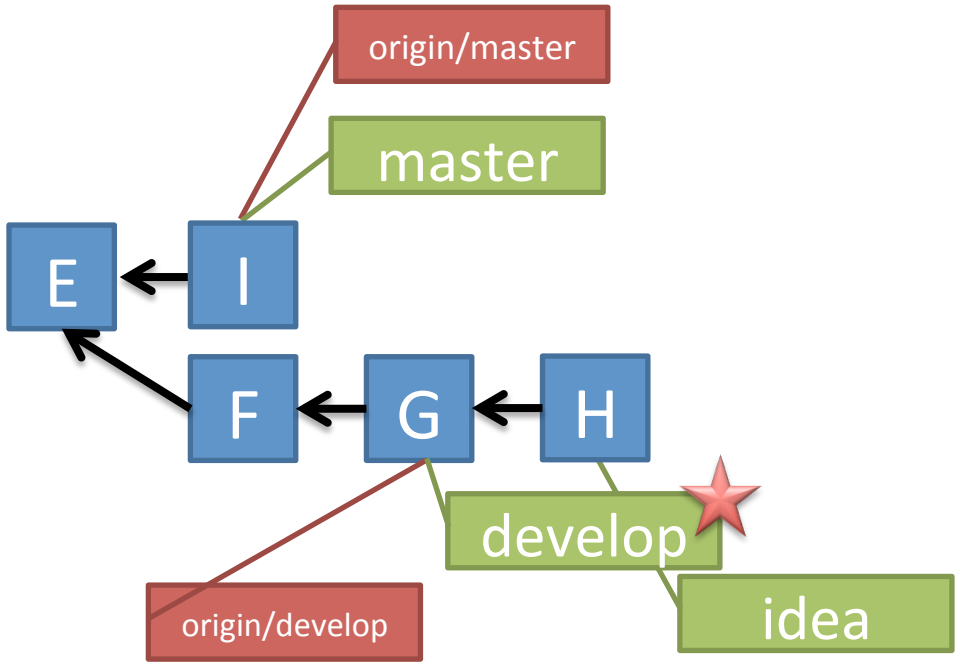


Branches Illustrated



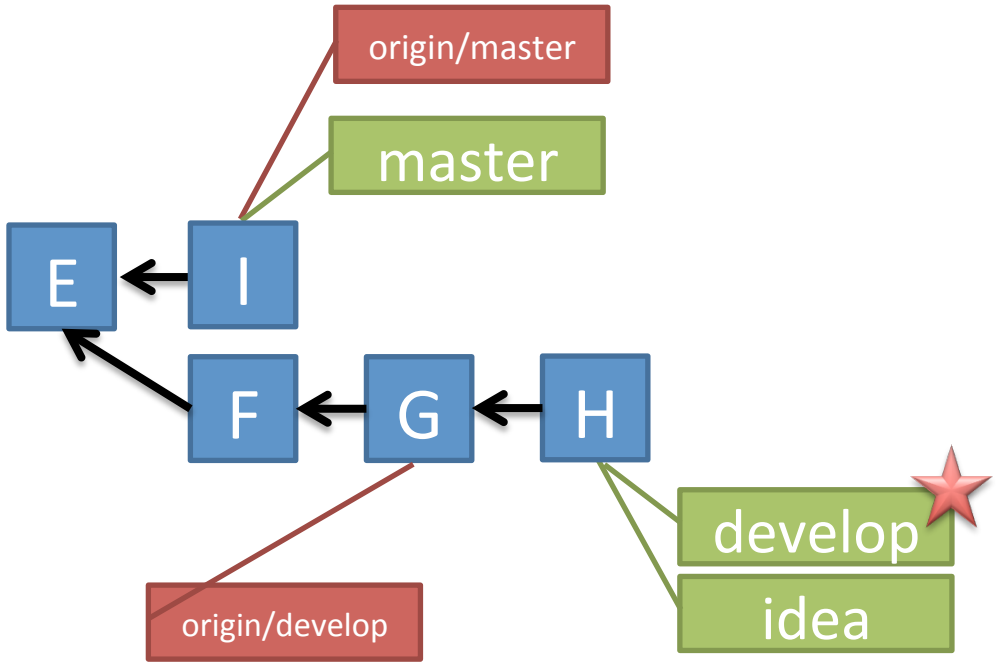

```
> git pull (at least daily)
```

Branches Illustrated



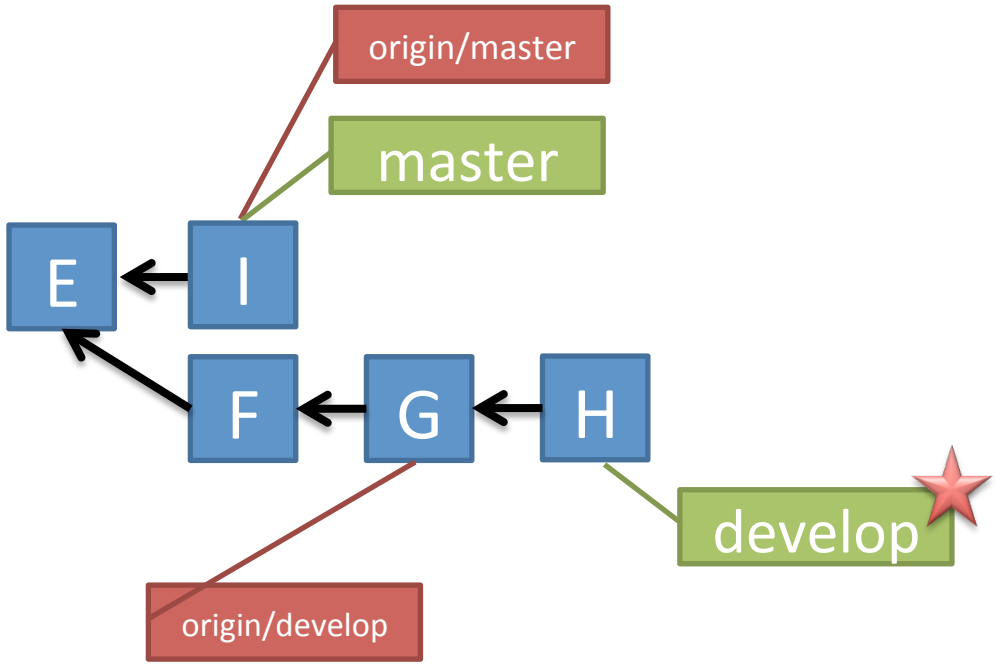

```
> git checkout develop
```

Branches Illustrated



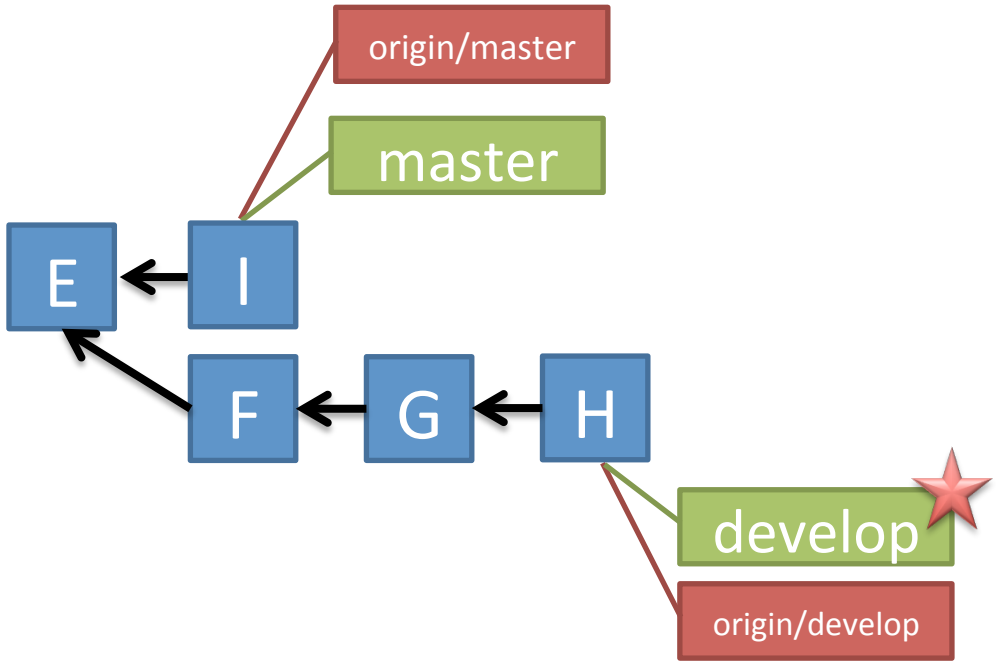

```
> git merge idea (fast forward merge)
```

Branches Illustrated



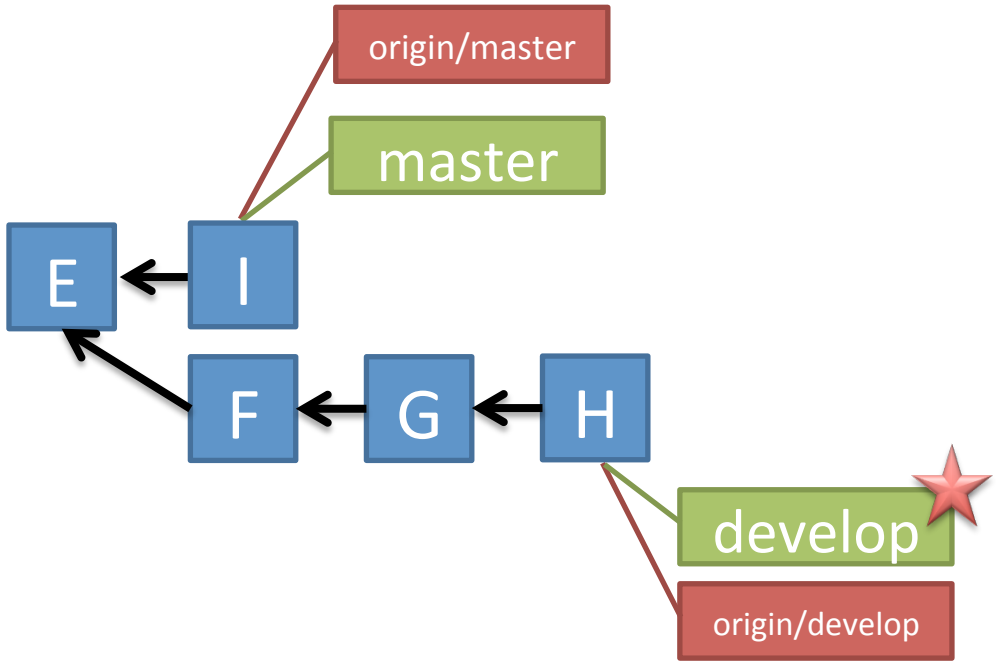

```
> git branch -d idea
```

Branches Illustrated



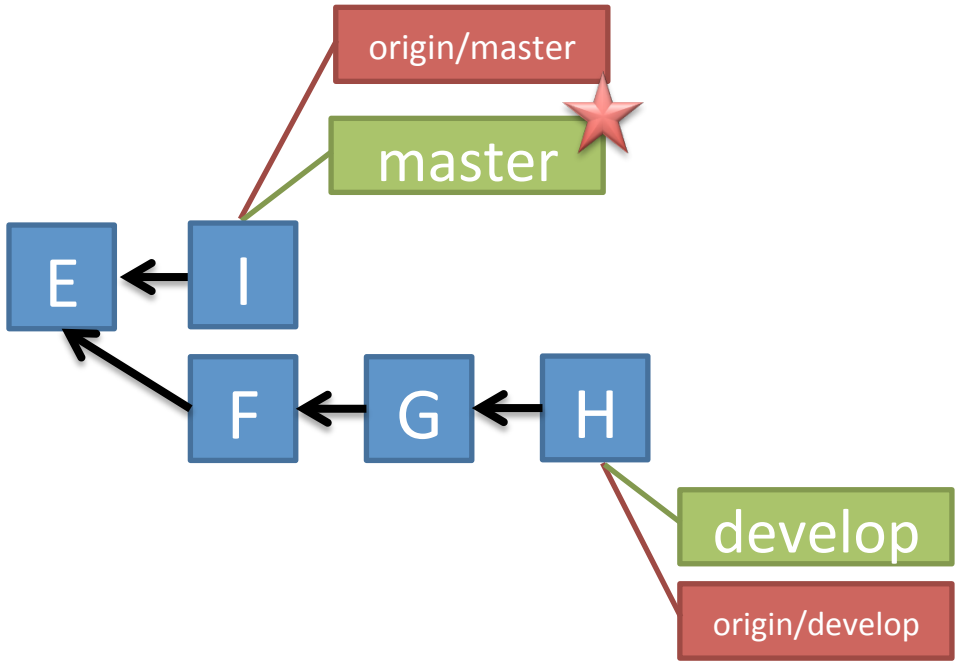

```
> git push origin develop
```

Merge Flow vs. Rebase Flow



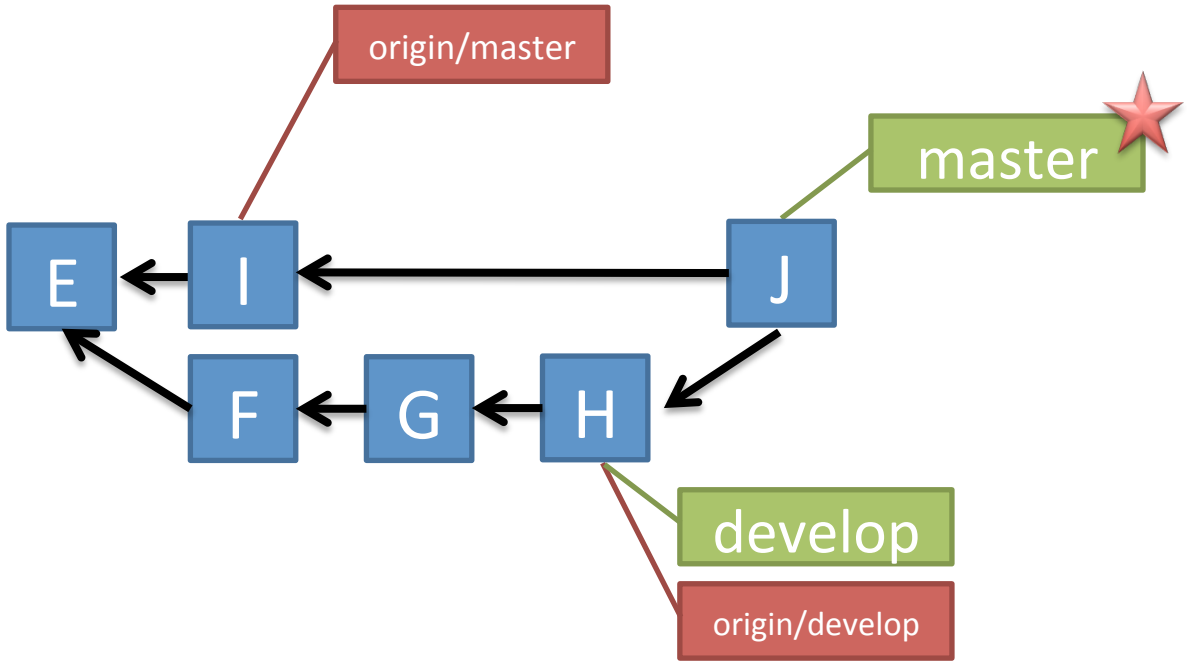

```
> git push origin develop
```

Branches Illustrated – Merge Flow



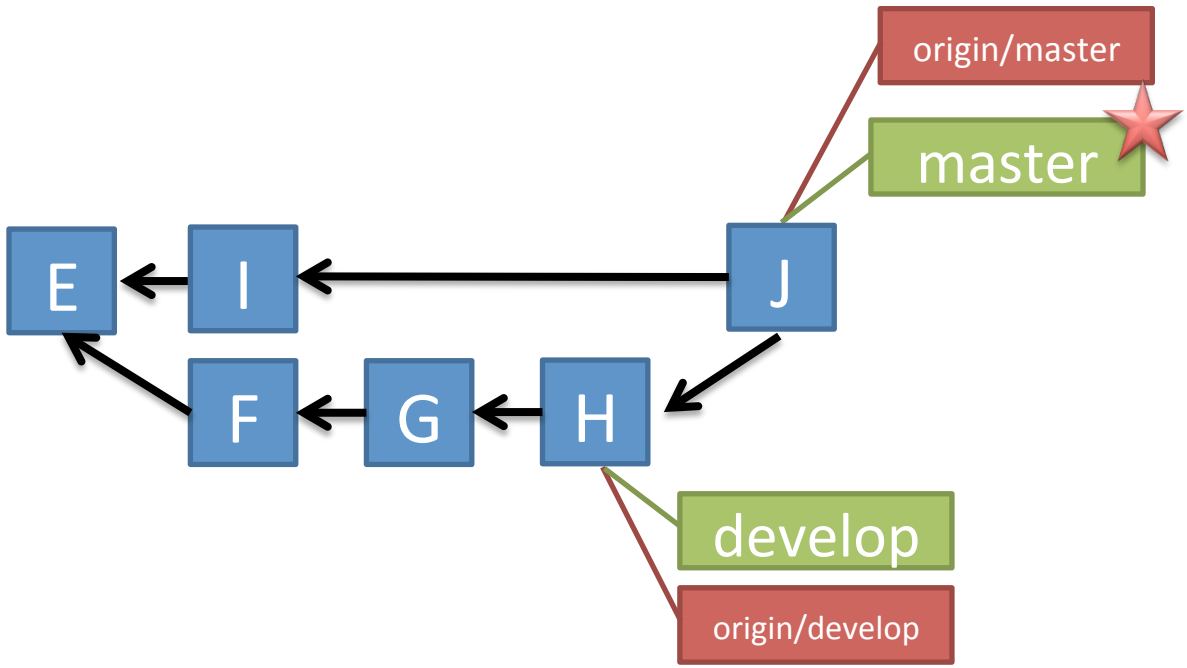

```
> git checkout master
```

Branches Illustrated – Merge Flow



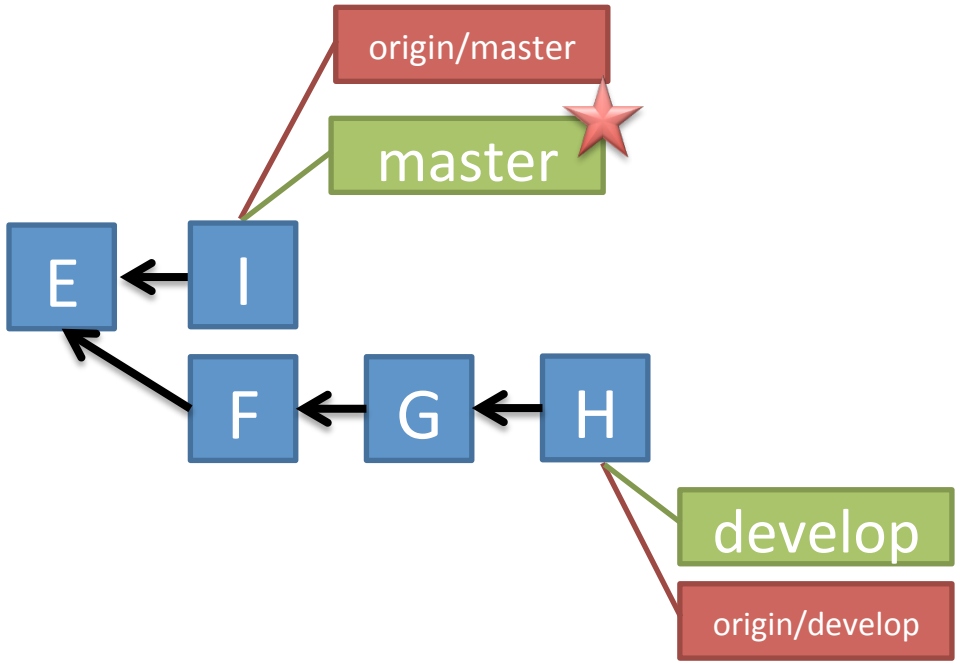

```
> git merge develop
```

Branches Illustrated – Merge Flow



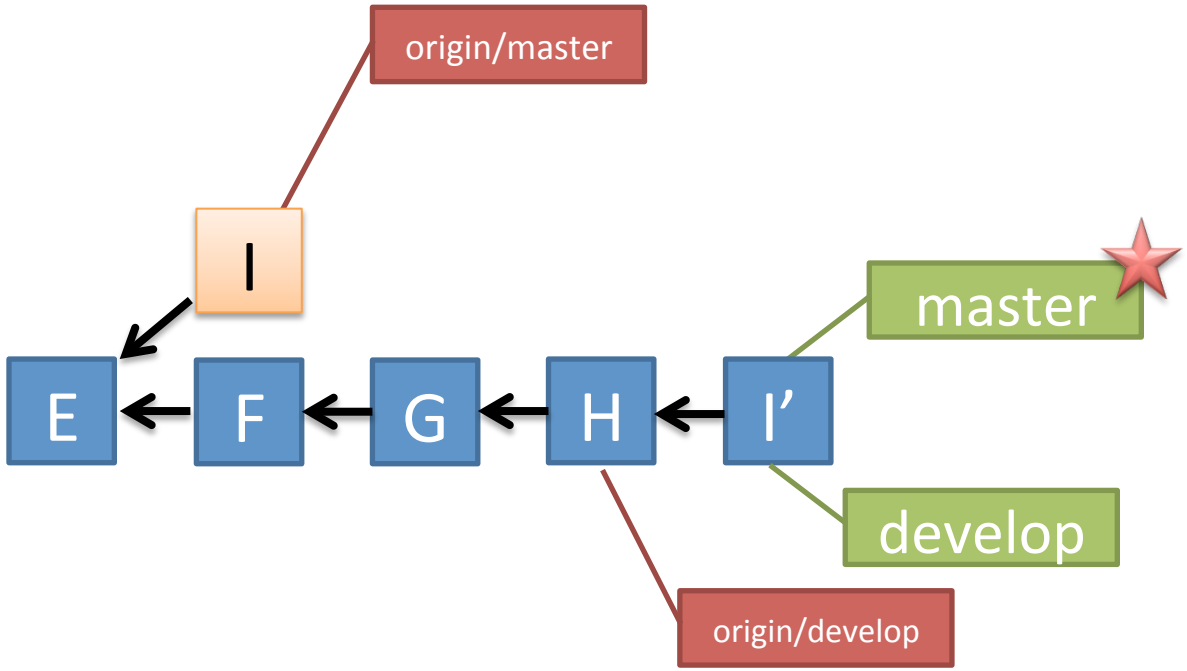

```
> git push origin
```

Branches Illustrated – Rebase Flow



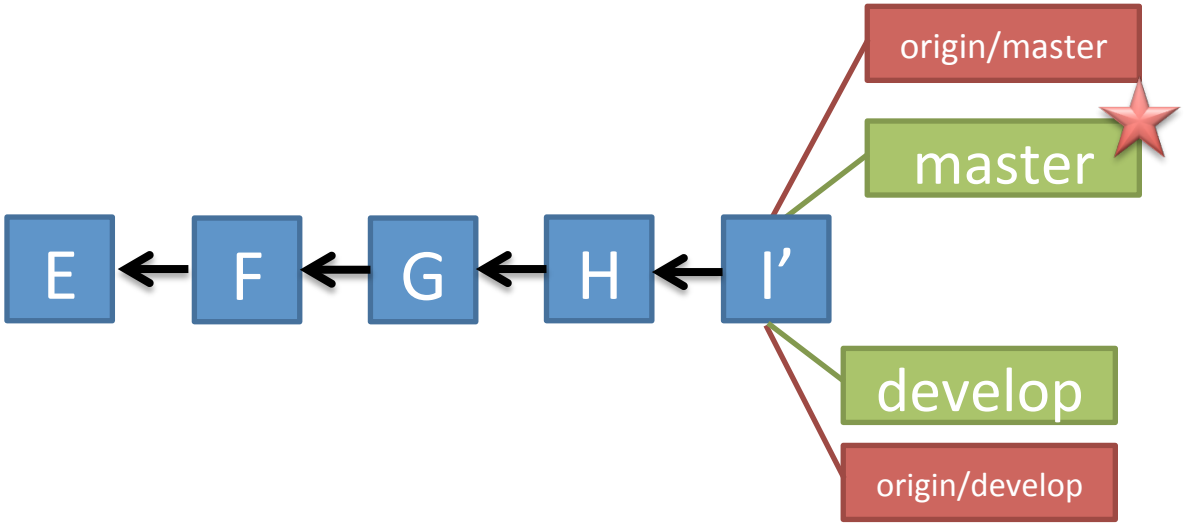

```
> git checkout master
```

Branches Illustrated – Rebase Flow



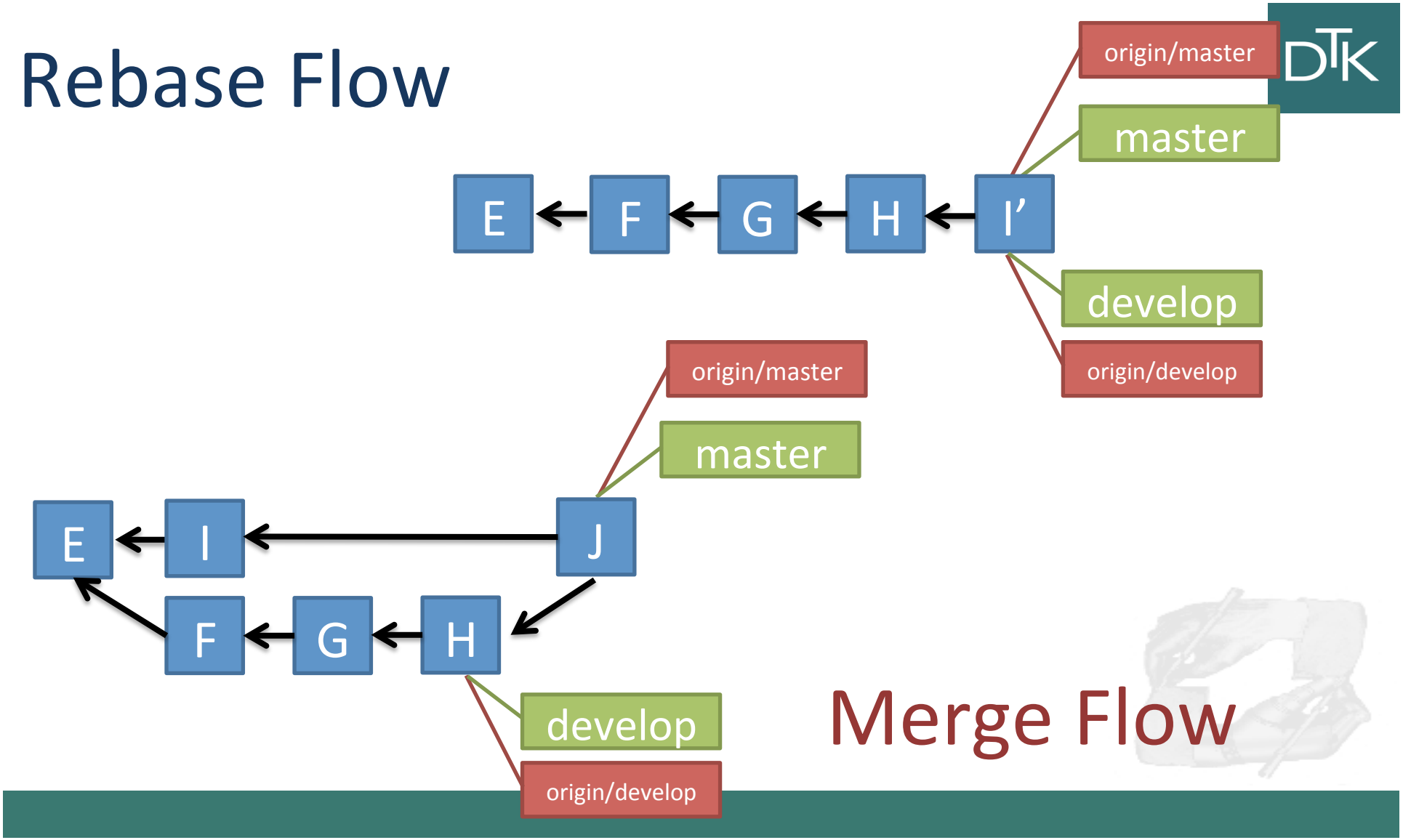

```
> git rebase develop
```

Branches Illustrated – Rebase Flow




```
> git push origin
```

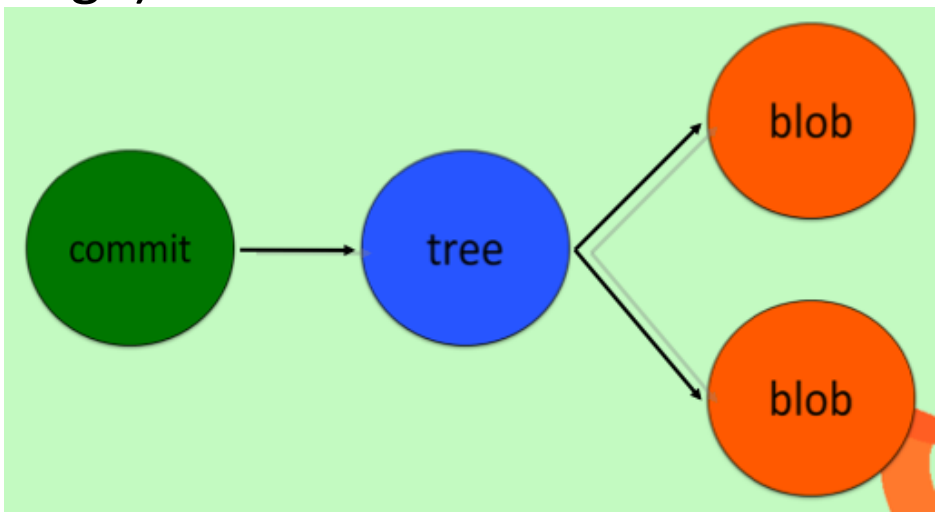
Rebase Flow



Merge Flow

How Git stores data

- Git stores the content of each file in the tracking history
- Each time we do a commit it is made a copy of the file.
- However the content of each file is subject to revision for conflicts (merge).



Git best practices for code collaboration

- When to commit?
  - Source of major arguments (big changes vs small change)
  - Never put broken code on the master branch (test first!)
  - Try not to break things (don't do two weeks worth of work in one commit)
  - Always use a clear, concise commit message
  - Put more details in lines below, but always make the first line short
  - Describe the why; the what is clear in the change log
- When making giant changes, consider branches (we'll talk about these in a few slides)
- Oh, and make sure your name and email are right

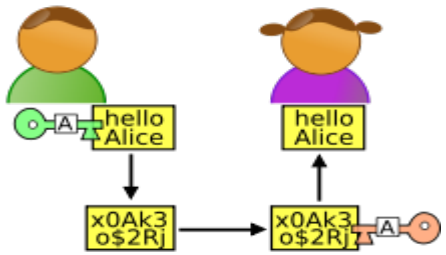
SUPPLEMENTAL

SSH

- Used to be most common transport for git
- Pros
  - Allows reads and writes
  - Authenticated network protocol
  - Secure: data transfer is encrypted and authenticated
  - Efficient: makes data as compact as possible
- Cons
  - No anonymous read-only access

Sidebar: What is SSH?

- SSH is a protocol used for secure network communication



Getting files from github

- Generate public/private keys (ssh-keygen)
- Distribute public keys (add key to github)
- Someone (github) sends secure “message” (files) – they encode with public key
- You receive the message/files – decode with private key (only you know)

Putting files on github

- Process is reversed to send files to github
- You have the github public key (see github_rsa.pub, in Documents and Settings/ Cyndi/.ssh on my machine)
- Use it to encode when sending
- github uses their private key to decode