Git : Part3
Branch Management

These slides were largely cut-and-pasted from http://excess.org/article/2008/07/ogre-git-tutorial/ , with some additions from other sources. I have deleted a lot from the cited tutorial, and recommend that you listen to the entire tutorial on line, if you can.

Branch Management

- Review
- Branch creation
- Merging
- Rebasing
- Putting it all together

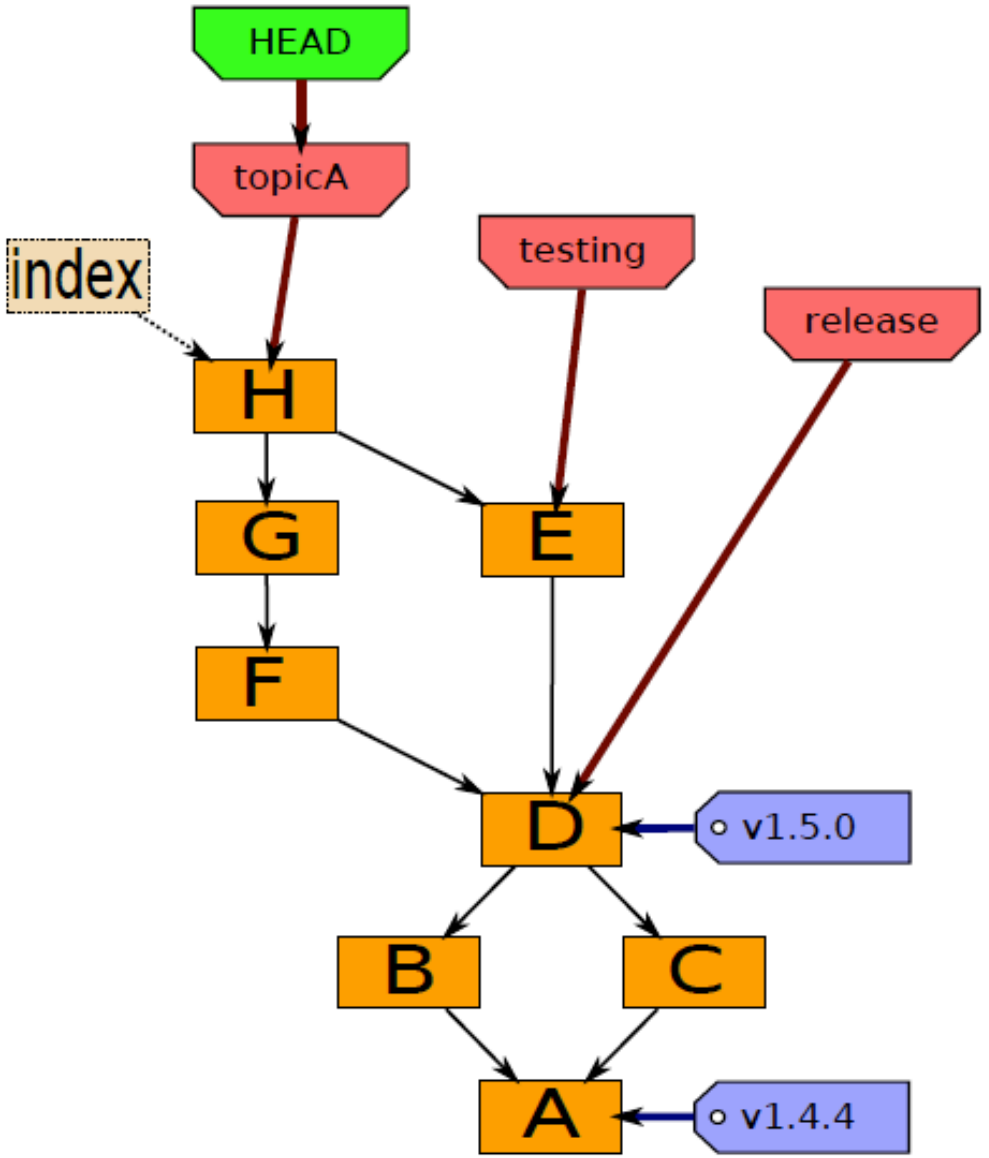
Review

Core git concepts

Git components

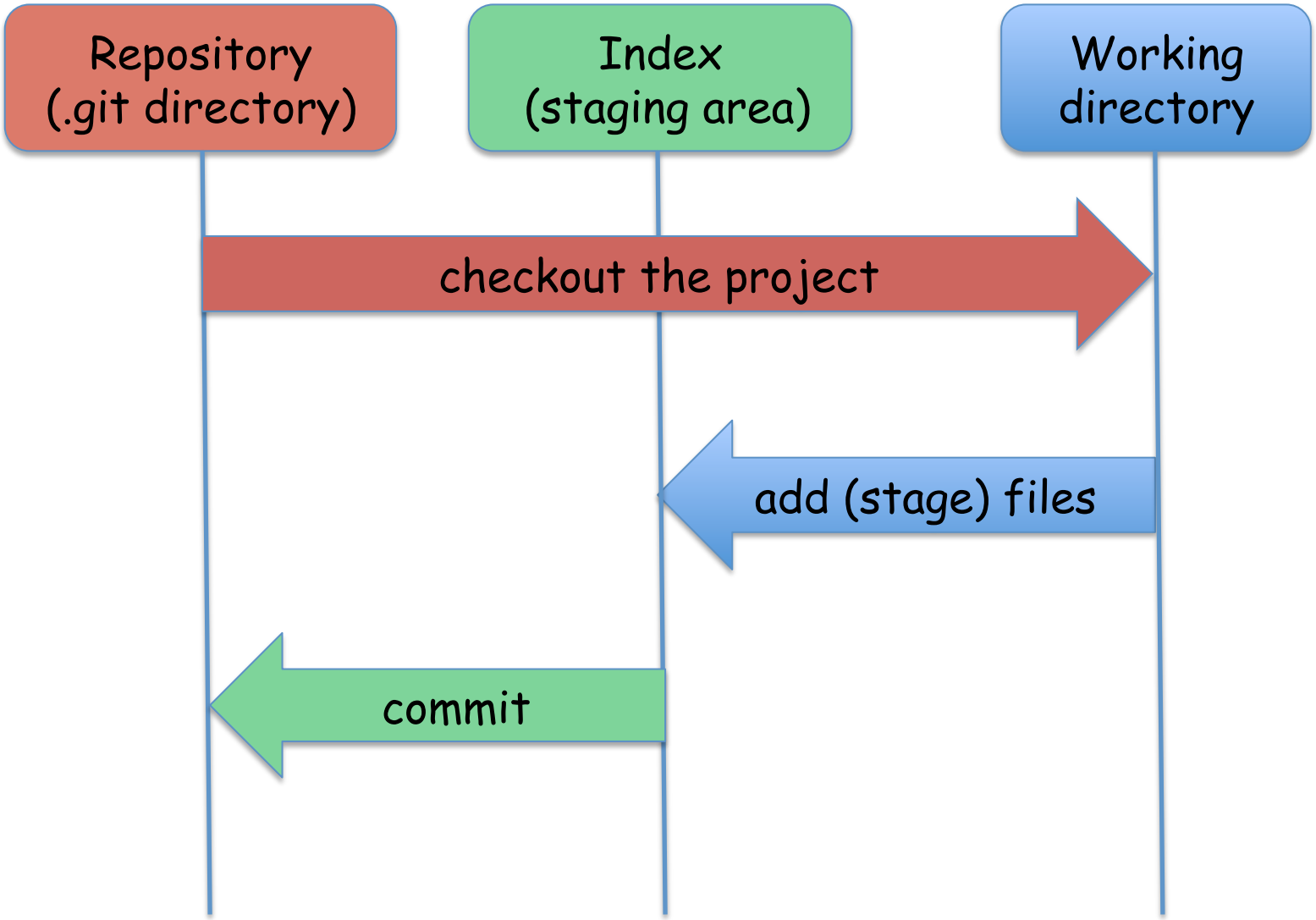
Index

- "staging area"
- what is to be committed

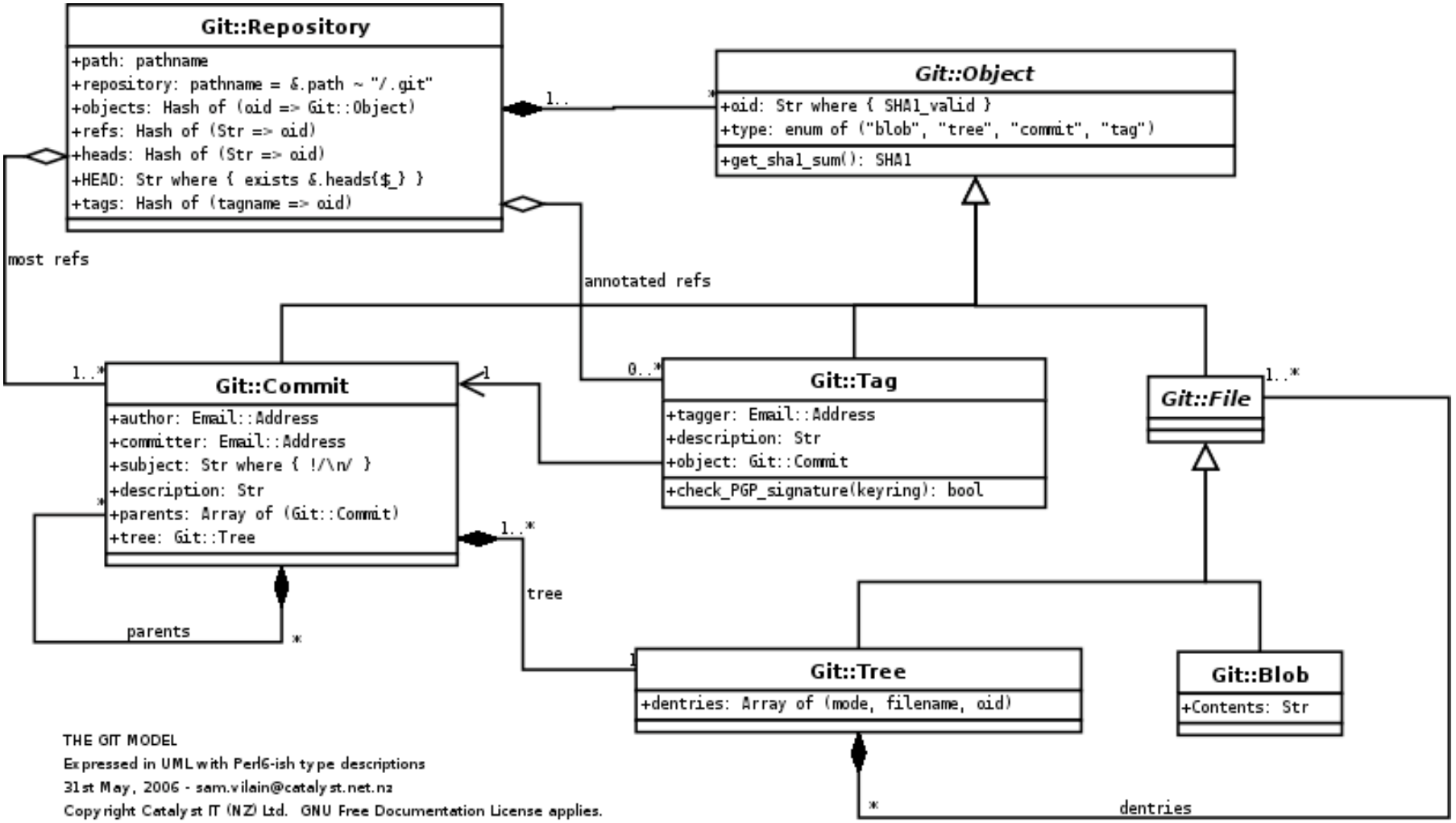


http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

Local Operations



Git object model



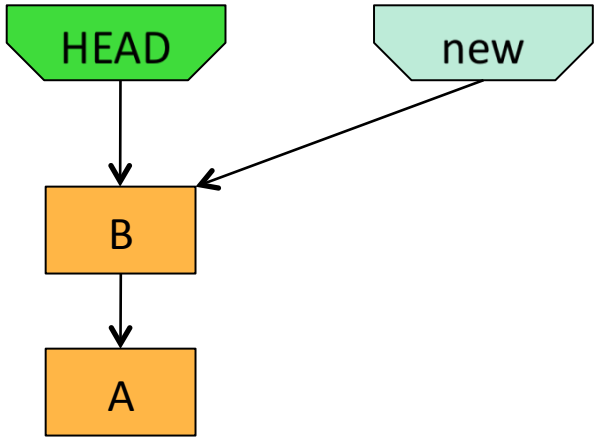
http://utsl.gen.nz/talks/git-svn/git-model.png

Branches

"alternate universes"

Creating branches

git branch <name> <commit>




```
git branch new HEAD
```

Local branches

To list them: `git branch –l`

```
    branch1
    branch2
  * master
```

Or look at files in `.git/refs/heads/`

Remote branches

To see them: `git branch –r`

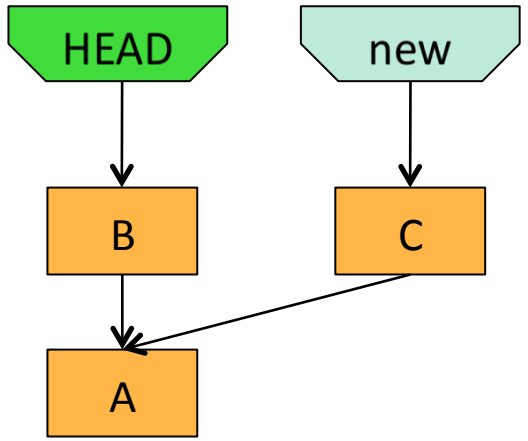
```
origin/HEAD -> origin/master
origin/master
origin/update
```

Or look at files in `.git/refs/remotes/`

Merging branches

when alternate universes collide

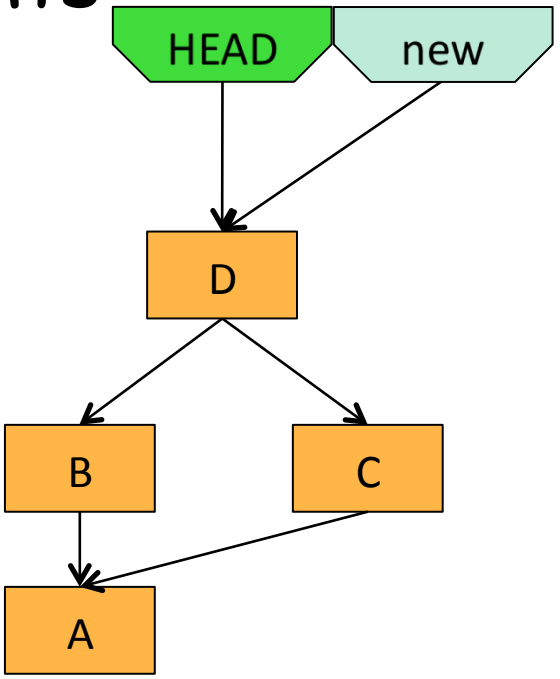
Merging



git merge <branch> …

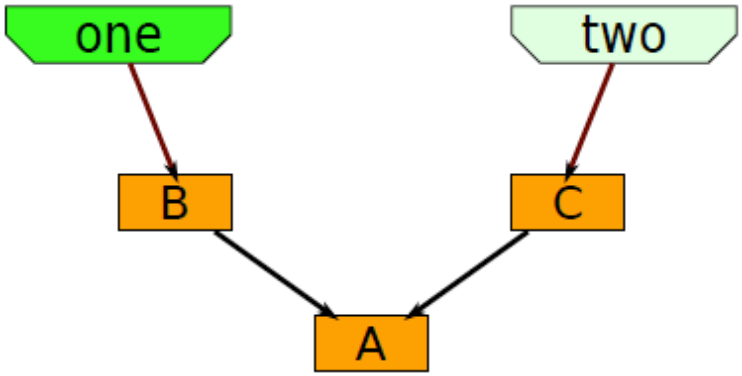
- joins branches
- creates commit with 2+ parents
- can cause conflicts

  requiring user intervention



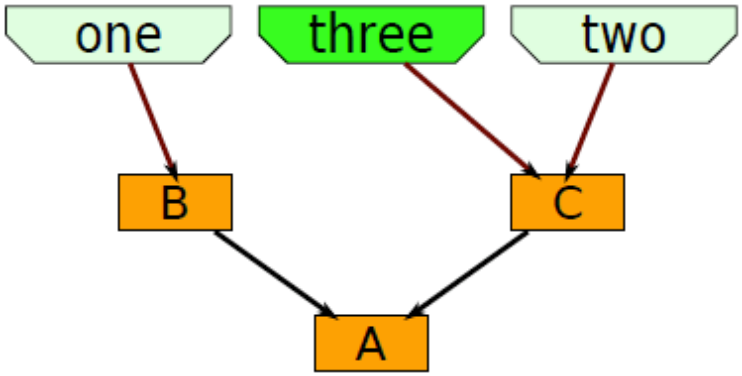

```
git merge new HEAD
```

Merge examples




```
git checkout –b three two
```

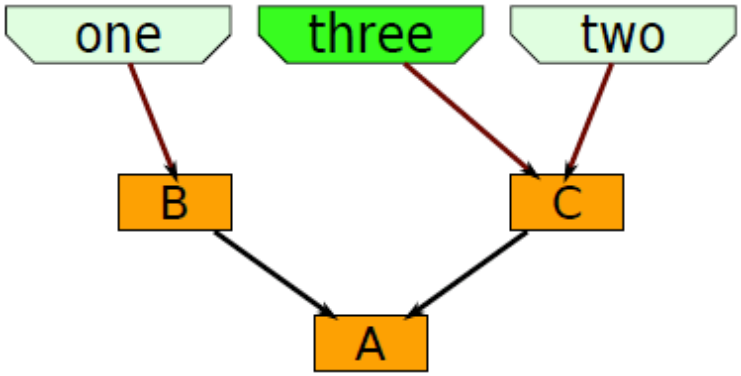
"checkout –b" creates a new branch and checks it out



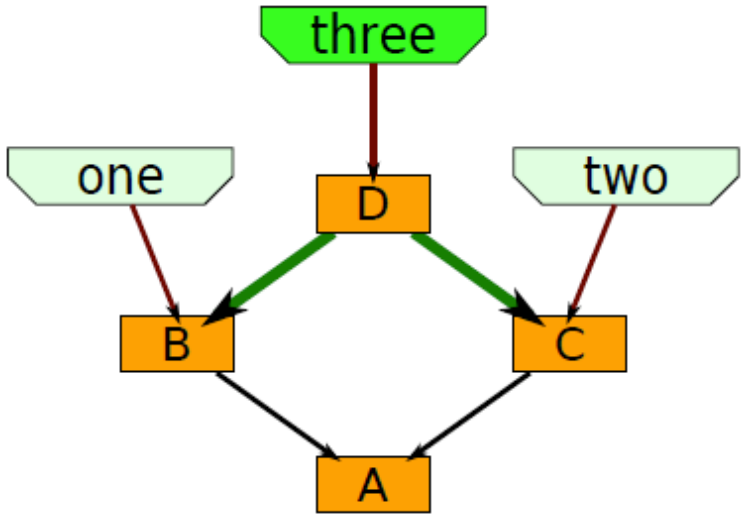
http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

2-way merge

```
git checkout –b three two
```

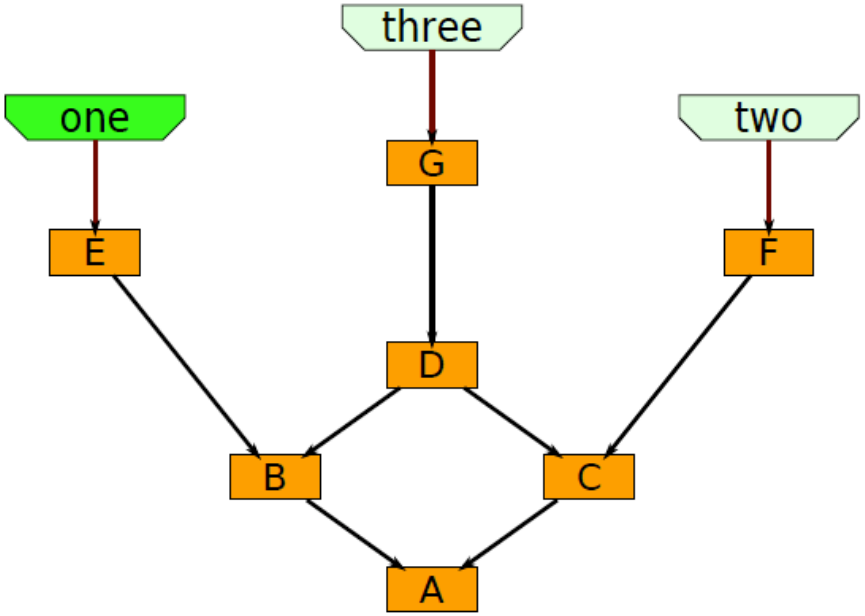


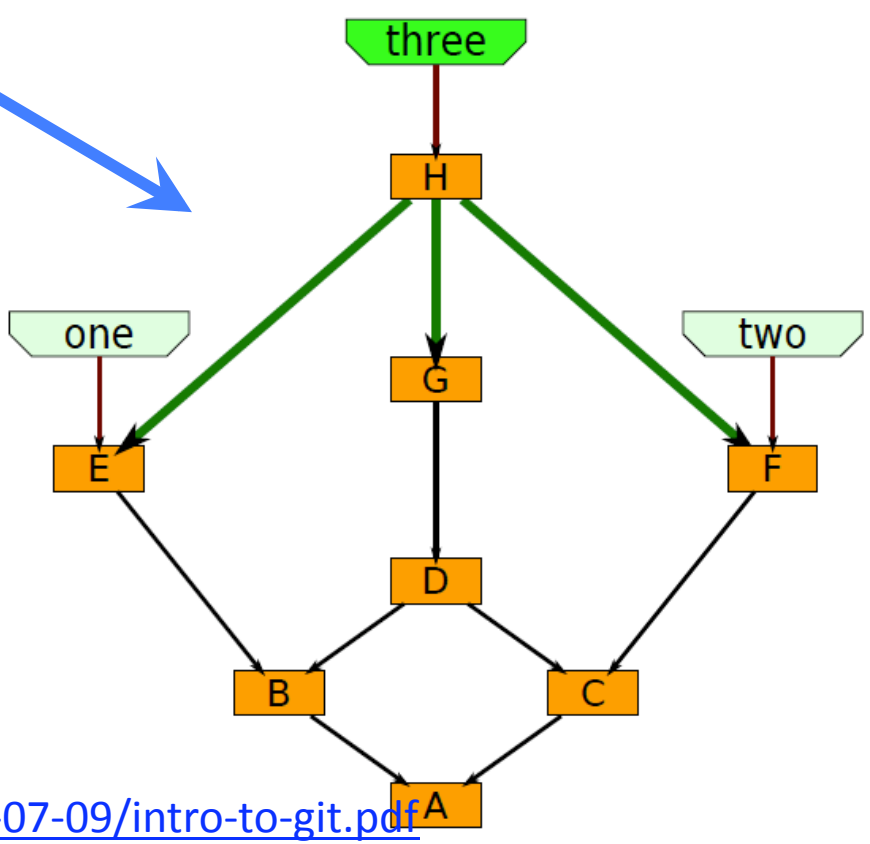

```
git merge one
```



http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

3-way merge



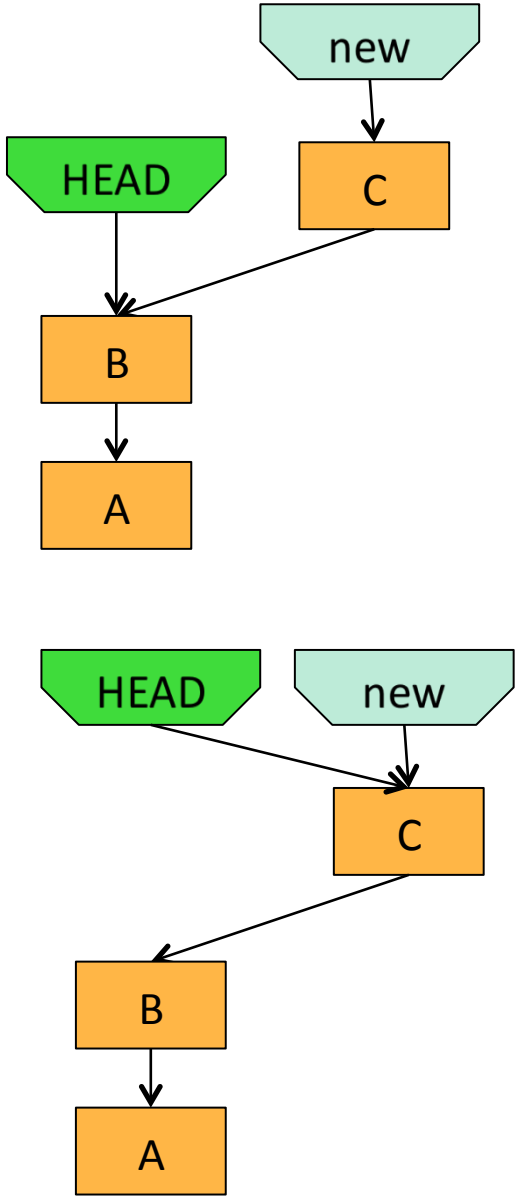



```
git checkout three
git merge one two
```

http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

Fast-forward merge

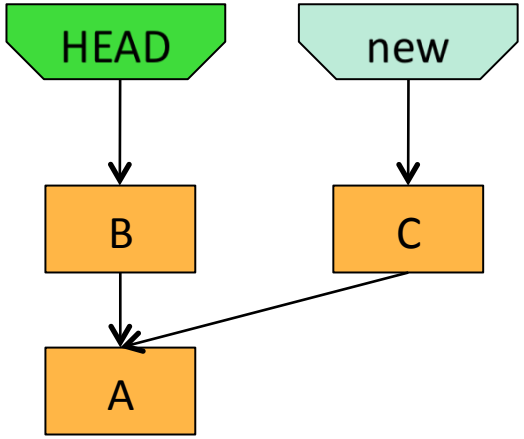
- Current head of the branch to which you are merging is an ancestor of the branch you are merging to it.
- The branch head is just moved to the newer commit.
- No merge commit object is created
  unless "-- no-ff" is specified

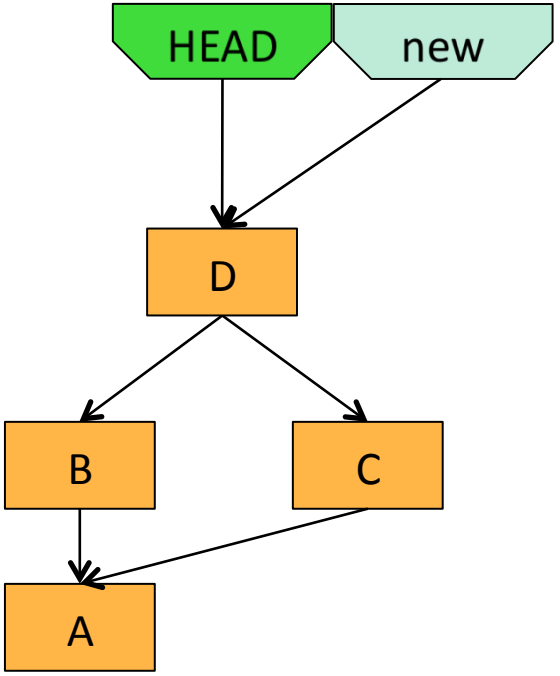



```
git merge new HEAD
```

True merge

- Not fast-forward
- New commit object must be created for new head
- 2 cases:
  - No overlapping changes are detected
    - Merge proceeds normally
  - Overlapping changes are detected
    Manual intervention is required





Beware of false security!

- Just because there are no overlapping changes does not mean the changes are semantically compatible.
  - It only means they do not modify the same region of the same file.
- So, unless the merge is a fast-forward, there is a good chance that your merge will break the software.
- This is the reason for following a discipline that forces all merges to be fast-forward.

Merge with conflicts

- HEAD pointer is unchanged
- MERGE_HEAD points to the other branch head
- Files that merged cleanly are updated in the index file and working tree
- 3 versions recorded for conflicting files:
  - Stage 1: common ancestor version
  - Stage 2: MERGE_HEAD version
  - Working tree: marked-up files (with <<< === >>>)
- No other changes are made
- You can start over with `git reset --merge`

How merge marks conflicts

```
Here are lines that are either unchanged from the common ancestor, or
cleanly resolved because only one side changed.
<<<<<<< yours:sample.txt
Conflict resolution is hard; let's go shopping.
=======
Git makes conflict resolution easy.
>>>>>>> theirs:sample.txt
And here is another line that is cleanly resolved or unmodified.
```

There is an alternate, 3-way, output option that also shows the common ancestor text.

Resolving merge conflicts

- Only two choices
  a. **Decide not to merge**: `git-reset --hard`
  b. Resolve the conflicts
- Resolution tools
  - **Use a mergetool: git mergetool**
    kdiff3,tkdiff, meld, xxdiff, emerge, vimdiff, gvimdiff, ecmerge, diffuse, tortoisemerge, opendiff, p4merge, araxis
  - Look at the diffs, and edit: git diff
  - …

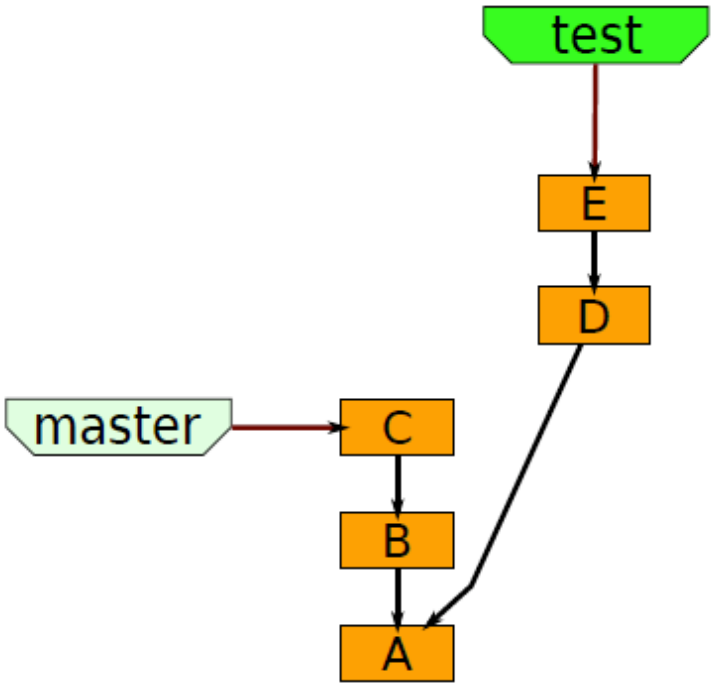
Rebasing

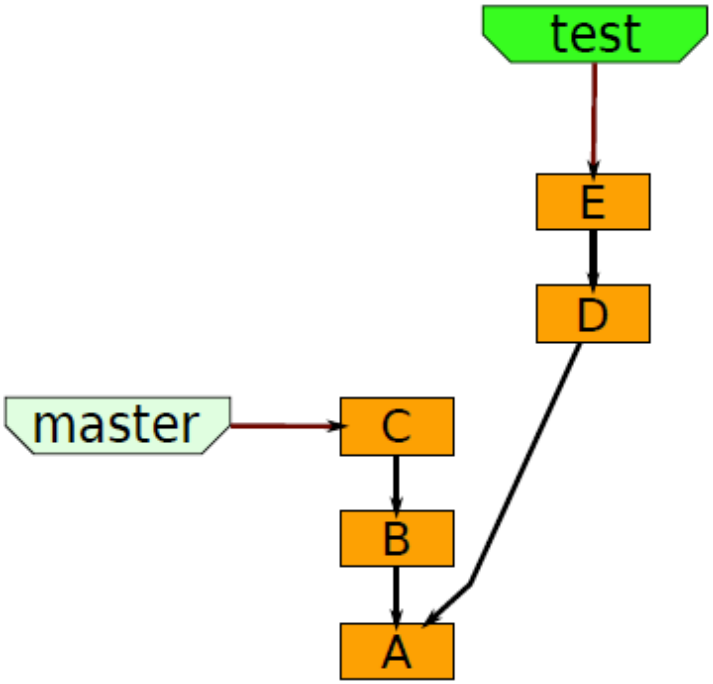
key to reducing the difficulty of merging

Rebase

- Contrasts to merge
  - Merge joins two branches
  - Rebase preserves branches
- Rolls changes from one branch into the other
  - Changes now are now relative to newer baseline
  - Allows tracking changes to baseline while developing new branch
  - Prevents surprises later
  - Avoids conflicts with eventual merges
- Rebase frequently to avoid merge conflicts

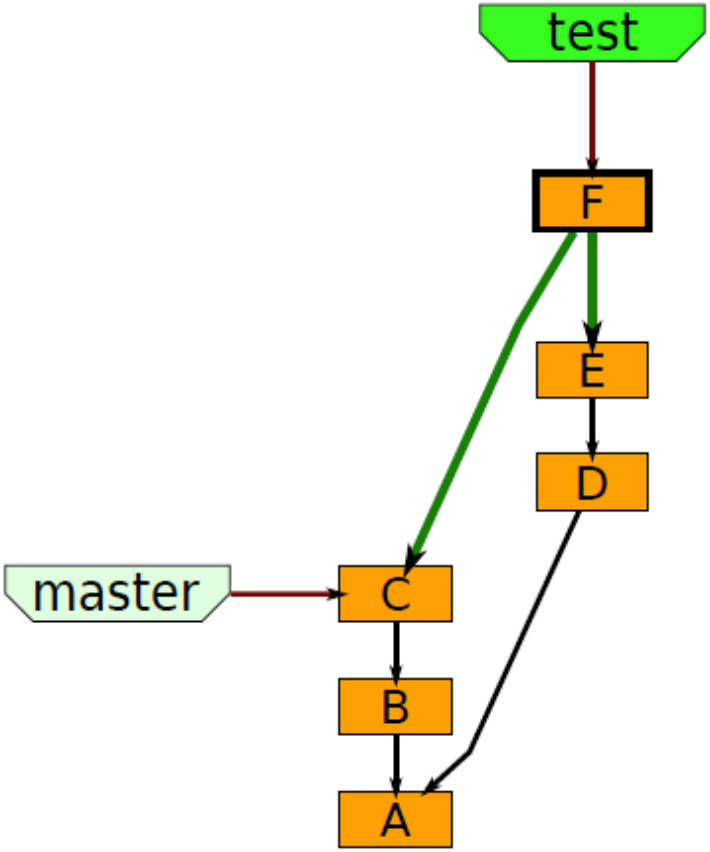
Merging vs. Rebasing

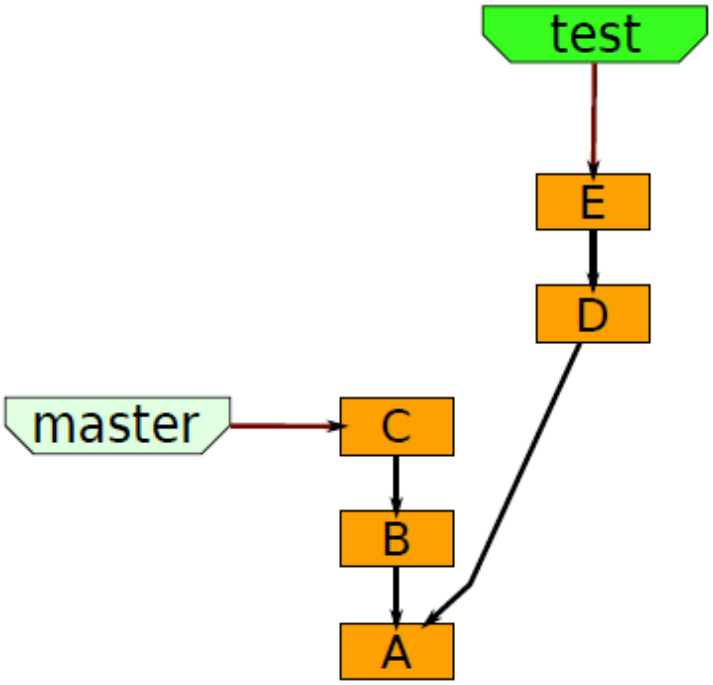




http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

Merging vs. Rebasing

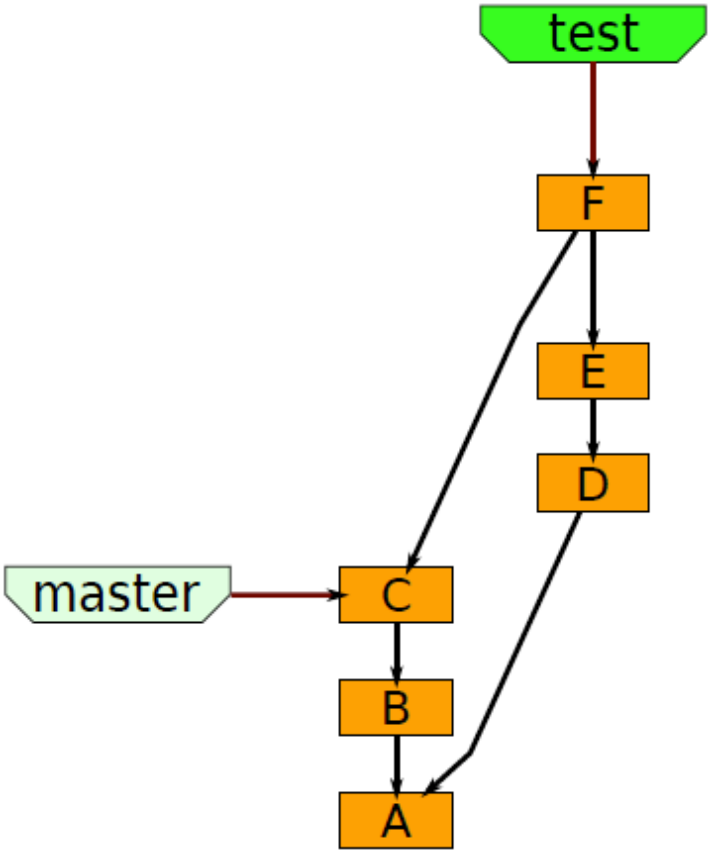


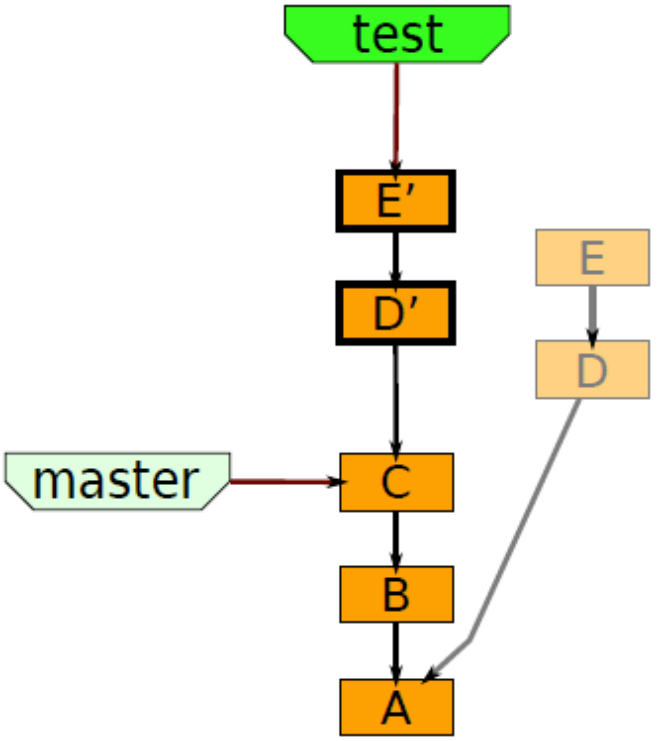



```
git merge master
```

http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

Merging vs. Rebasing

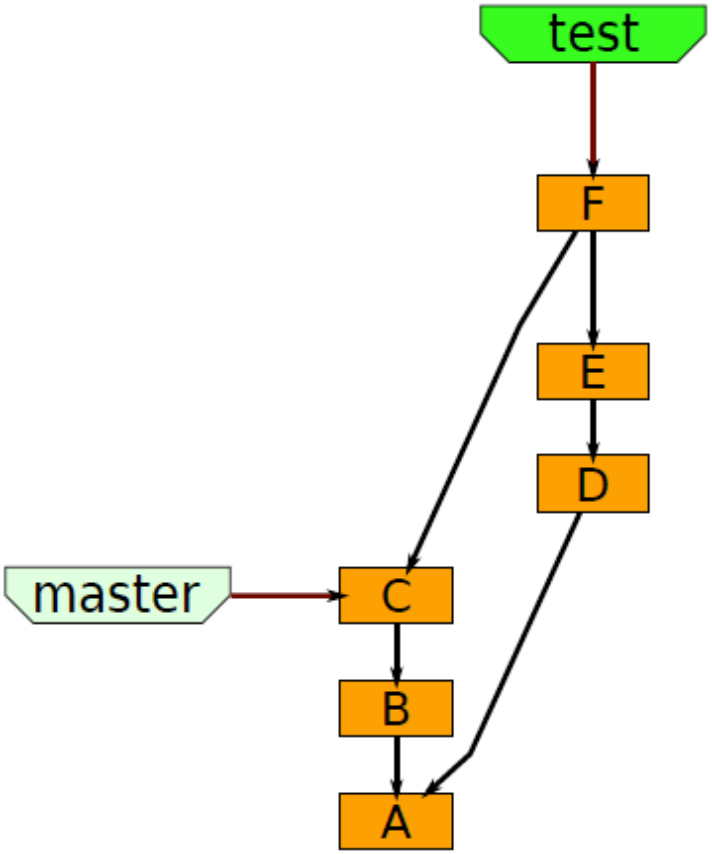


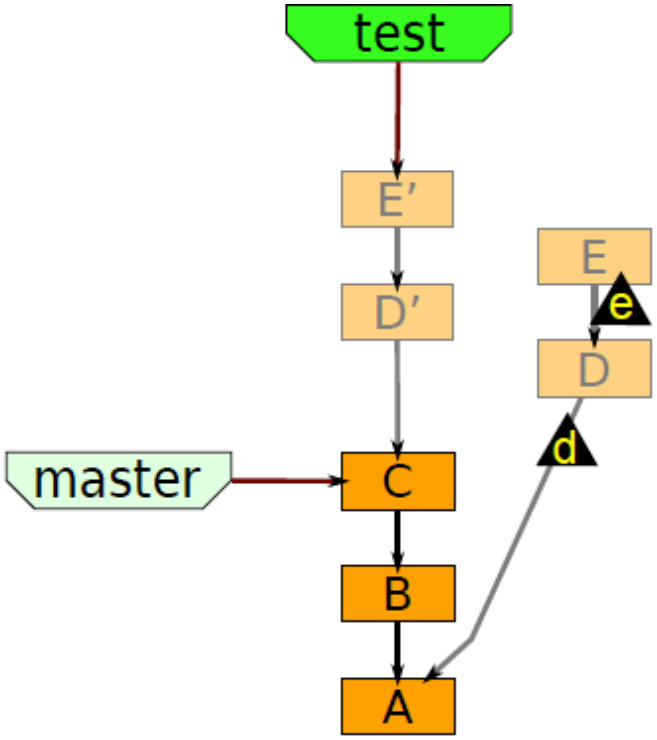



```
git rebase master
```

http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

Merging vs. Rebasing

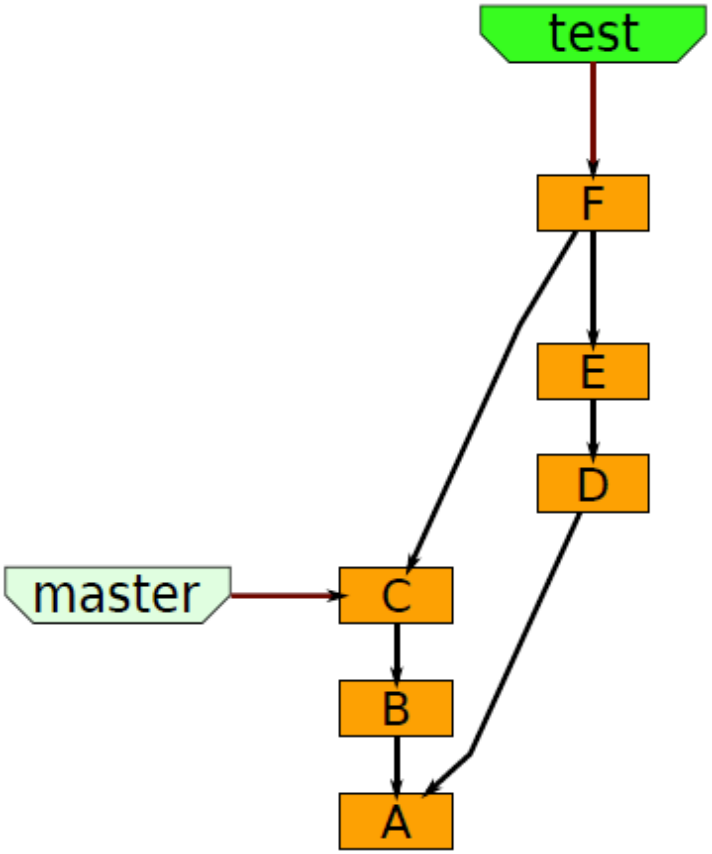


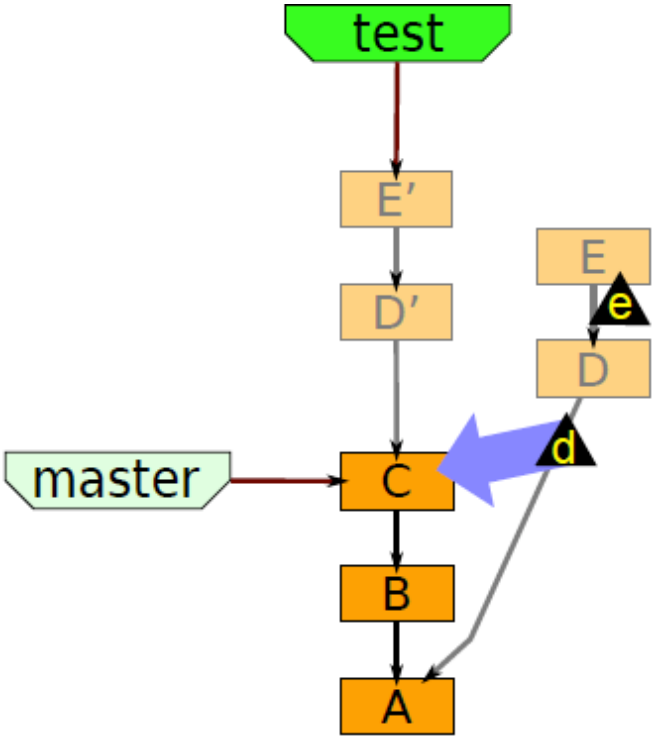


`git rebase master`

http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

Merging vs. Rebasing

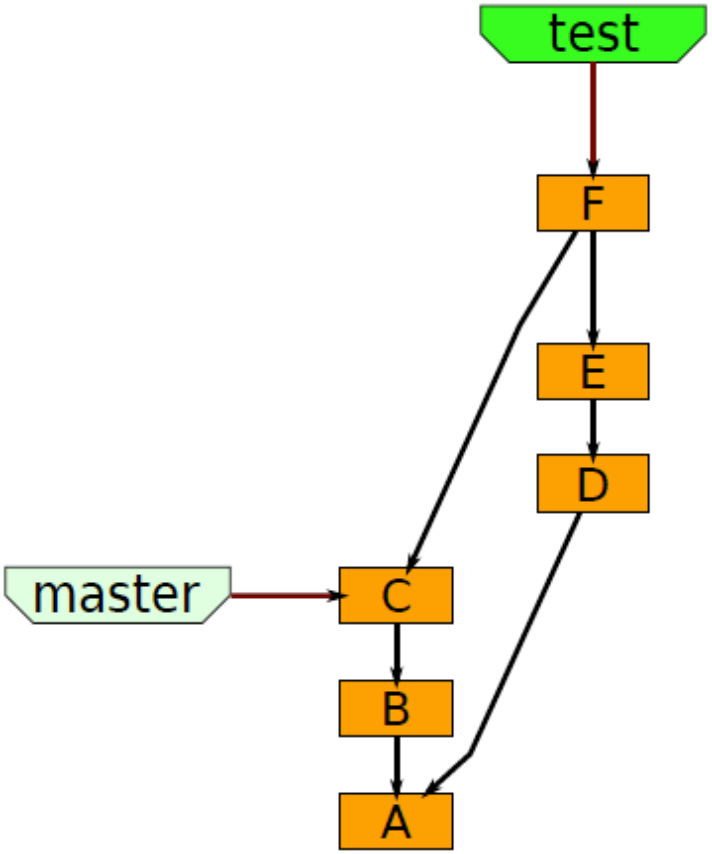


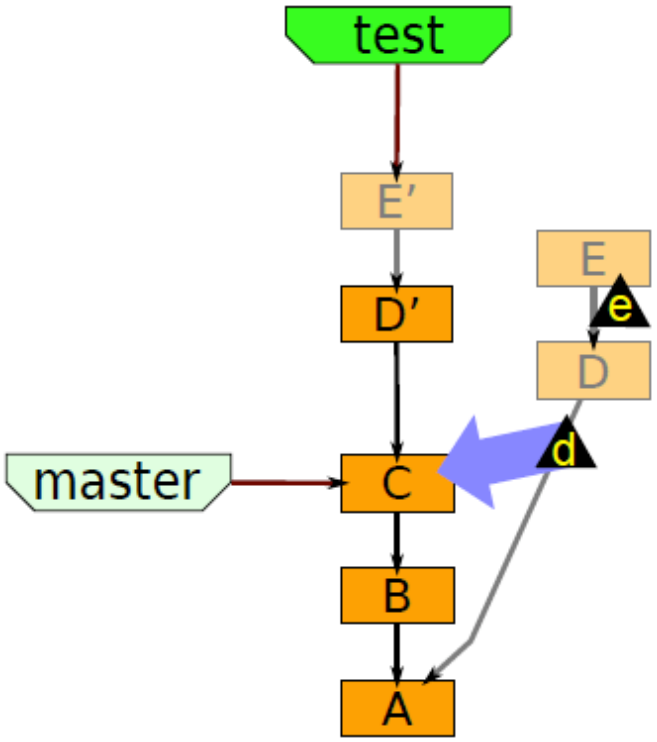


`git rebase master`

http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

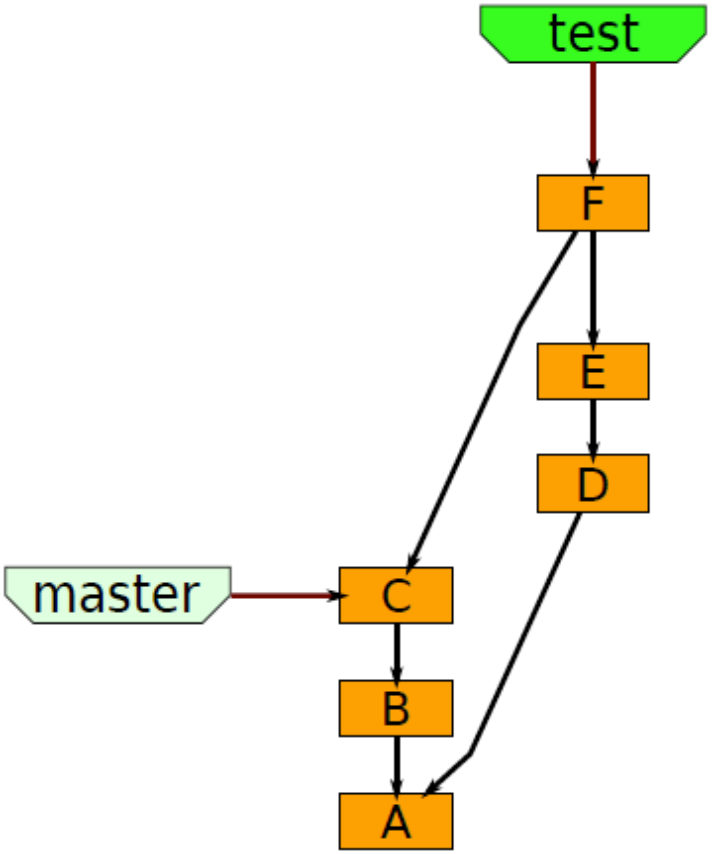
Merging vs. Rebasing

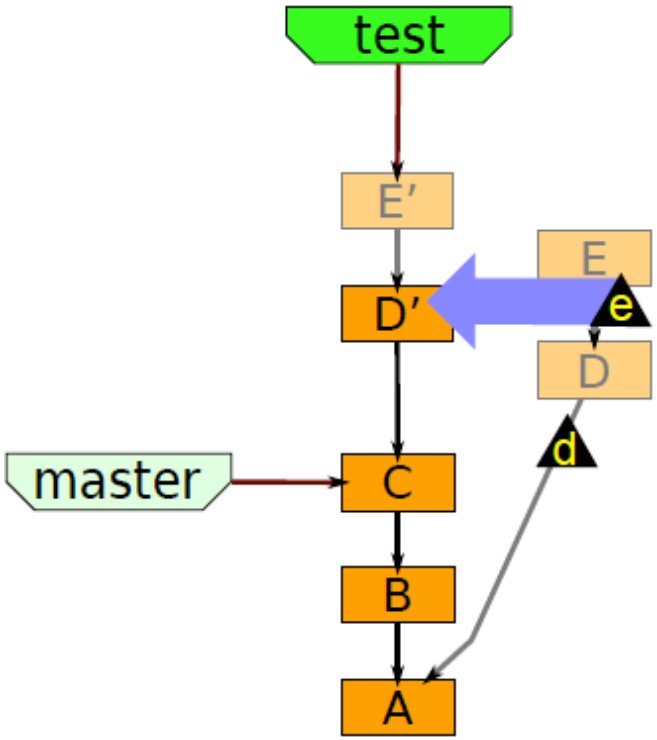




`git rebase master`

http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

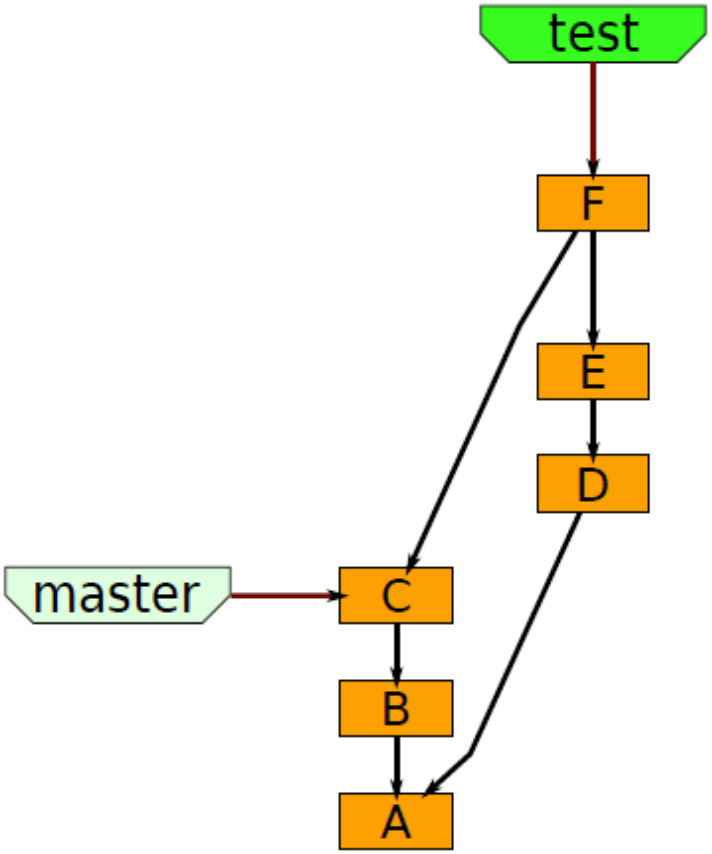


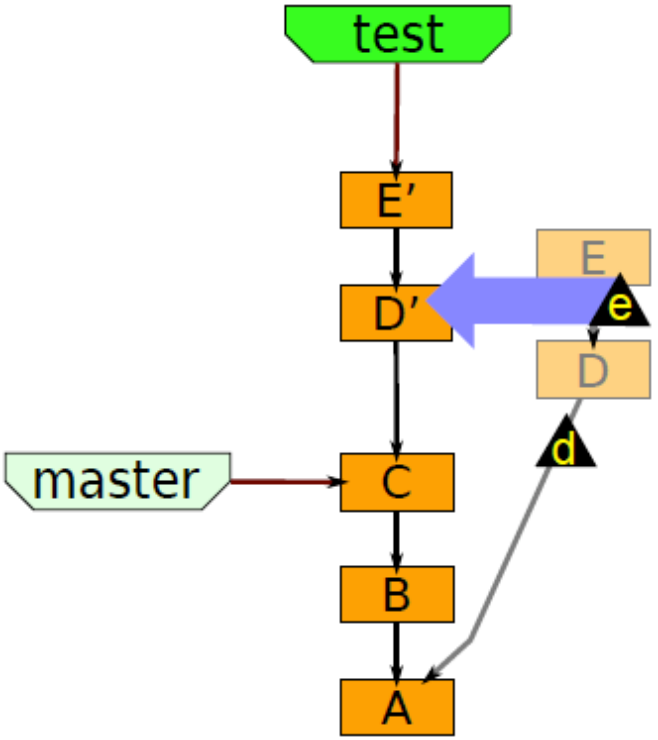


`git rebase master`

http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

Merging vs. Rebasing

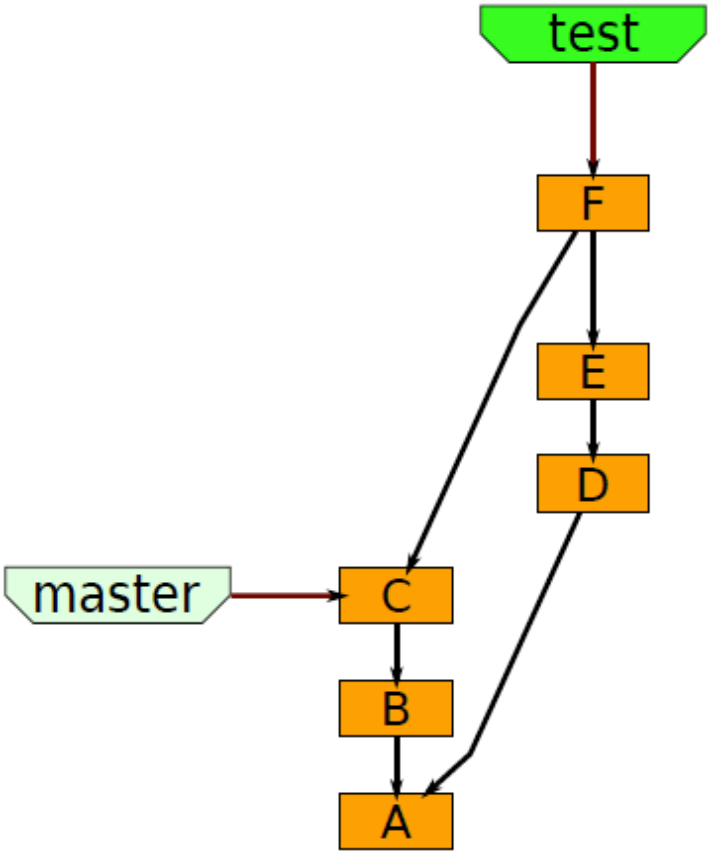


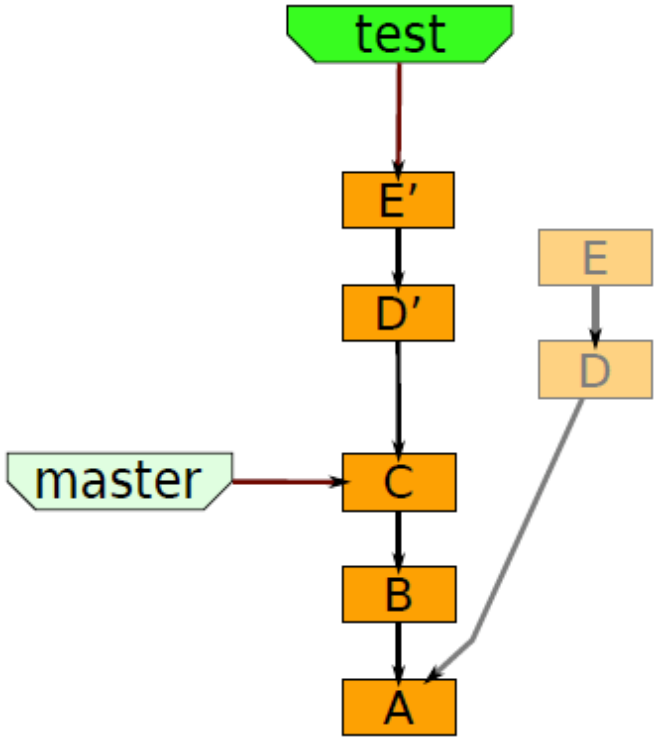


`git rebase master`

http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

Merging vs. Rebasing

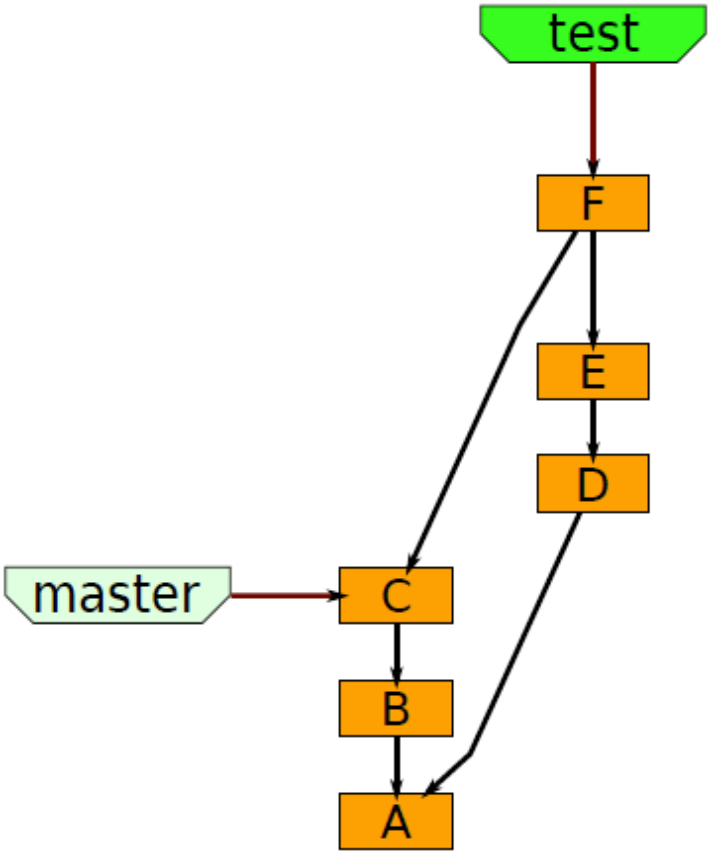


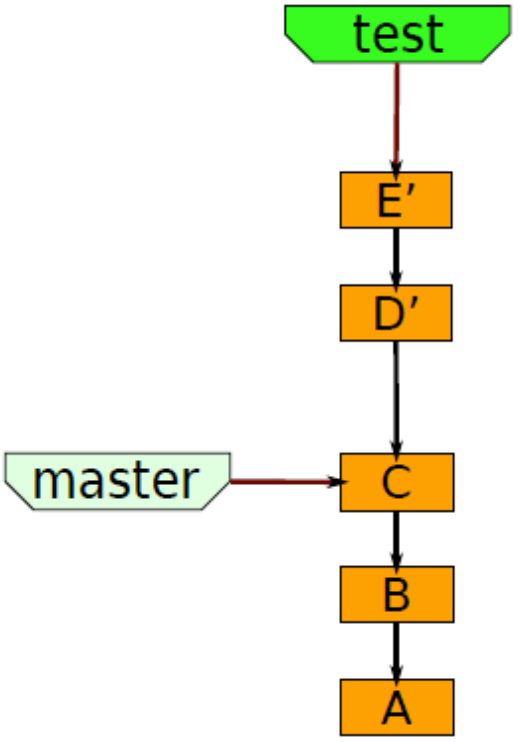



```
git rebase master
```

http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

Merging vs. Rebasing





`git rebase master`

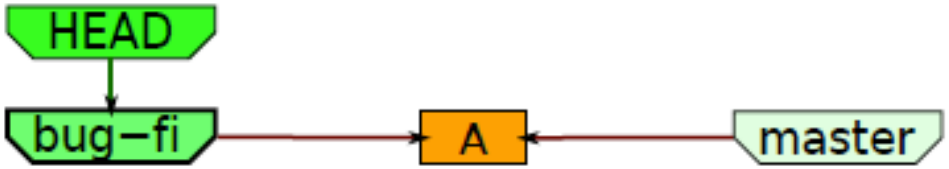
http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

Putting it all together

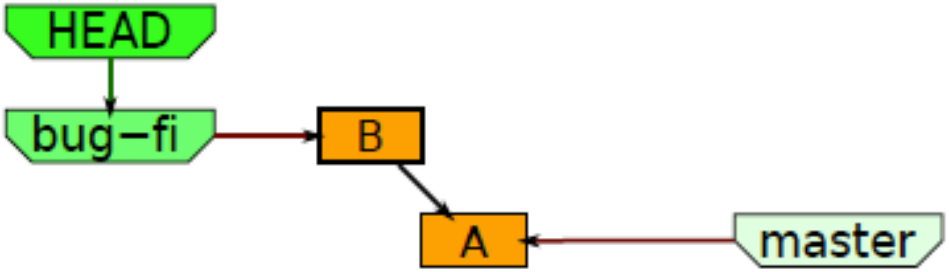
A user story showing how rebase
is used with branch and merge,
to reduce pain of merging.

Starting out to fix a bug


```
git checkout –b bug-fix
```

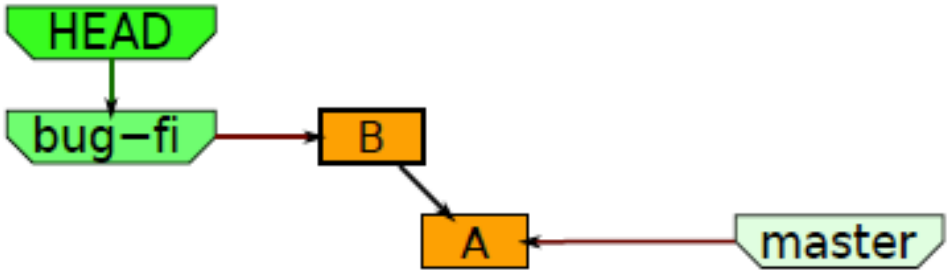



```
git commit –a –m”B”
```

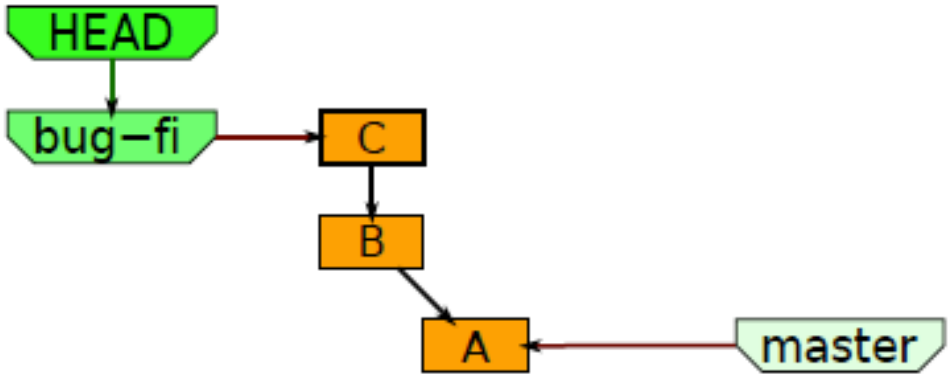


http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

Continue making changes

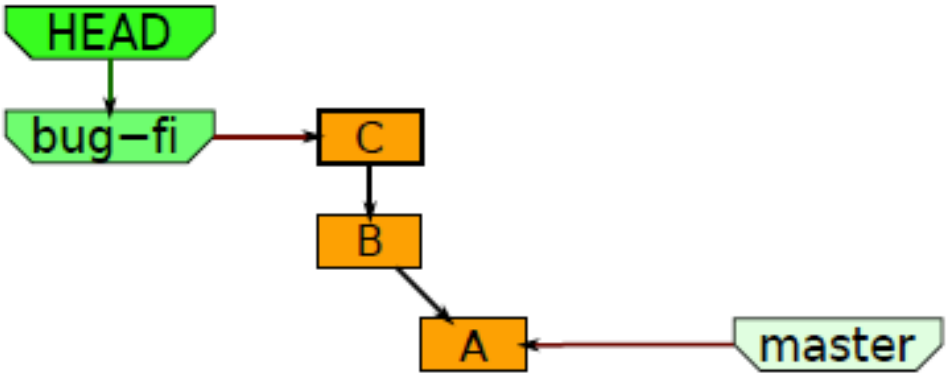



```
git commit –a –m"C" –b
```

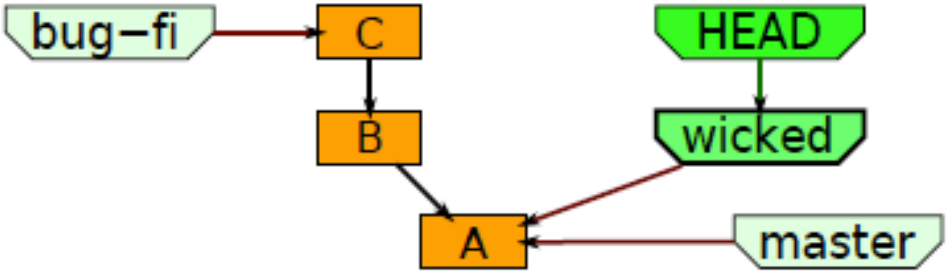


http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

Decide to try out a "wicked" idea.



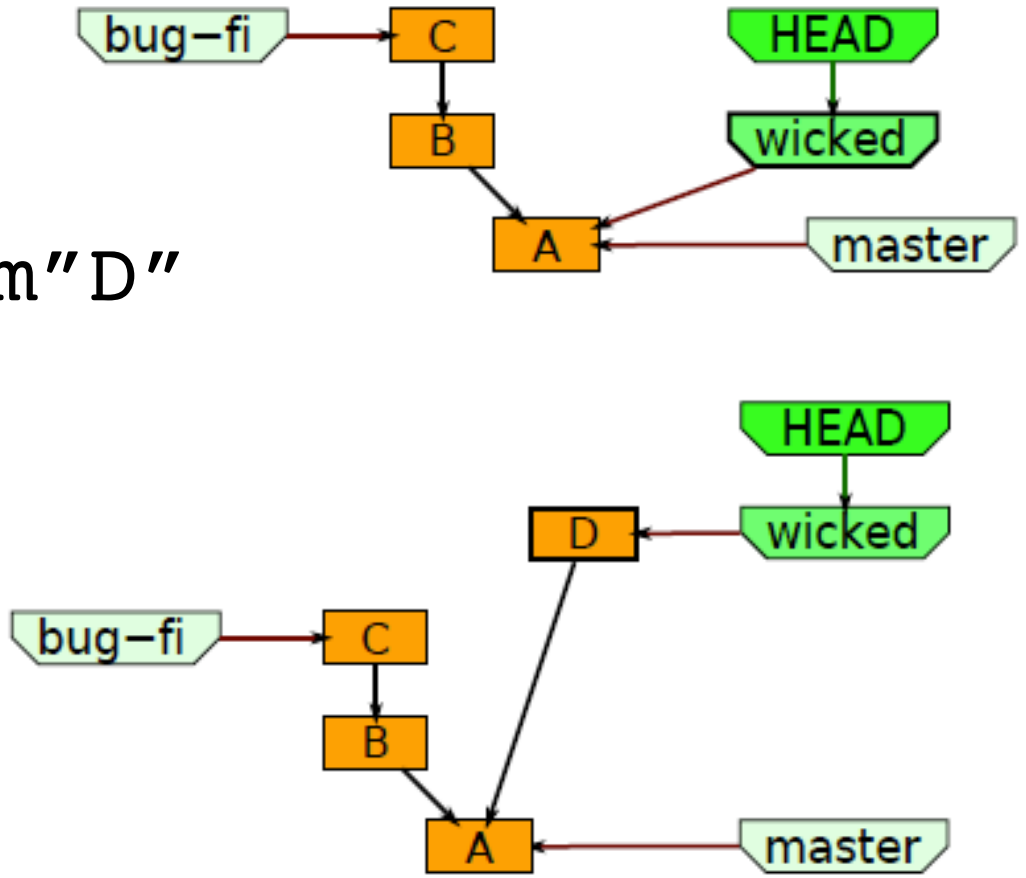

```
git checkout –b wicked master
```



http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

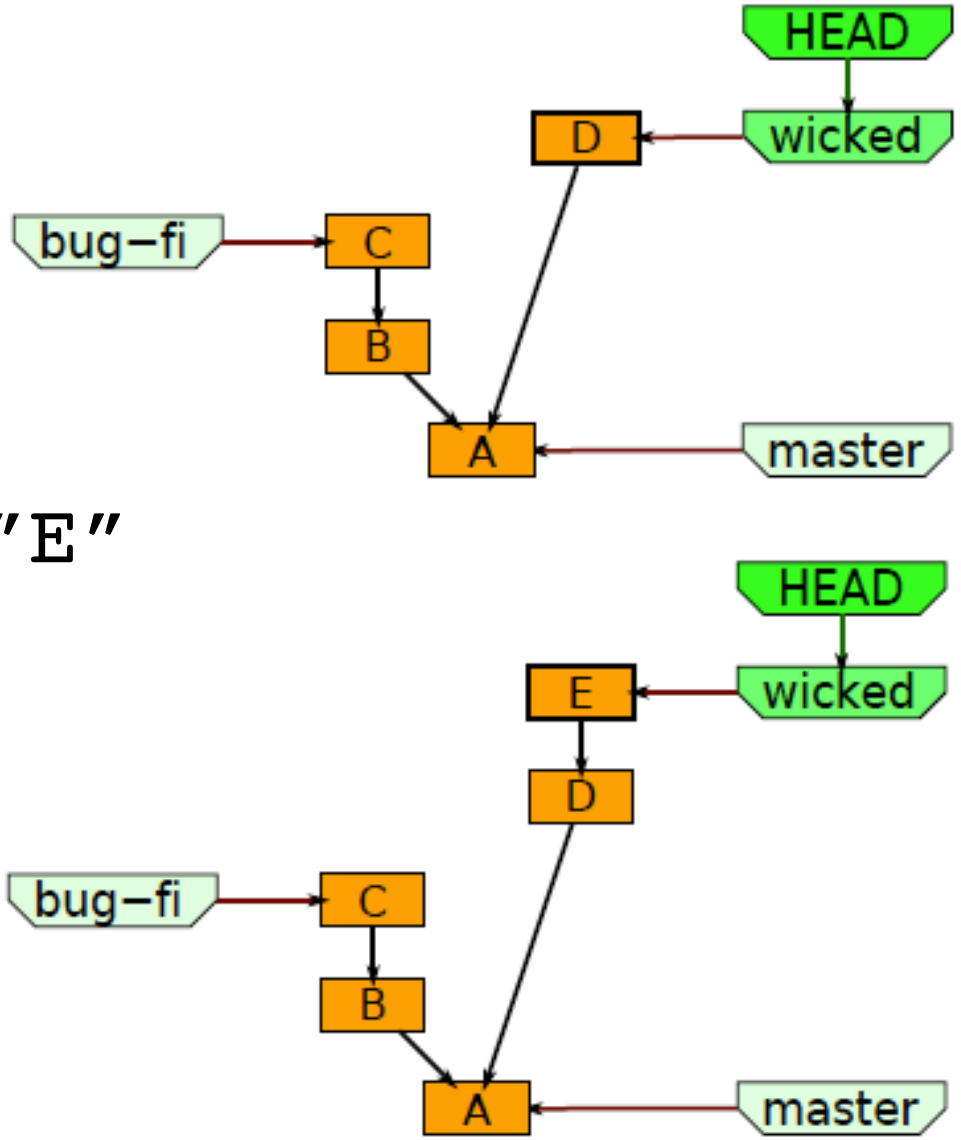
Work on the wicked branch

```
git commit –a –m"D"
```



http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

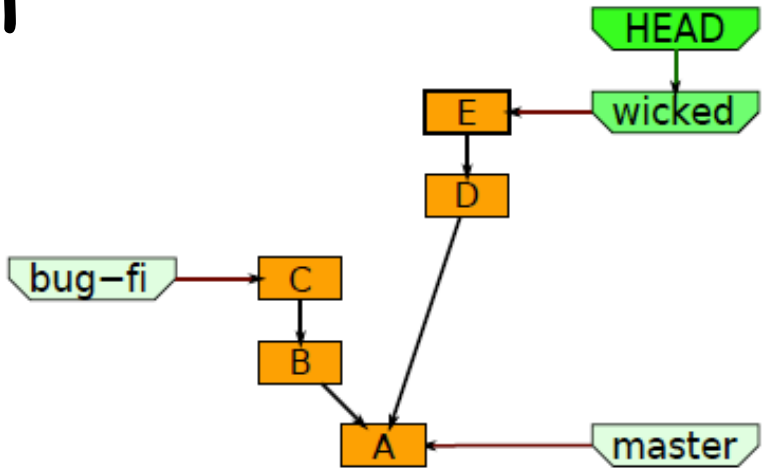
And some more



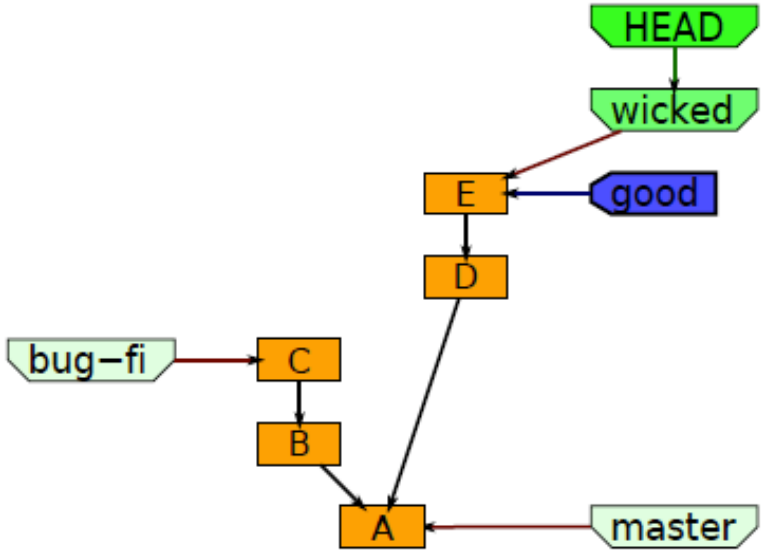

```
git commit –a –m"E"
```

http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

Tag a good point



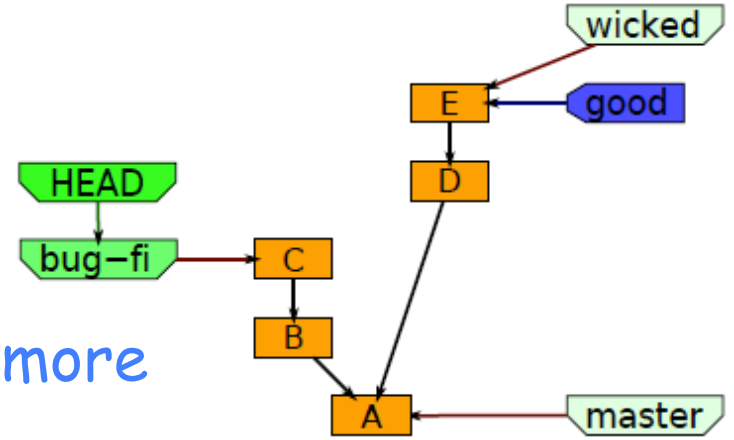

```
git tag –a –m"got somewhere" good
```



http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

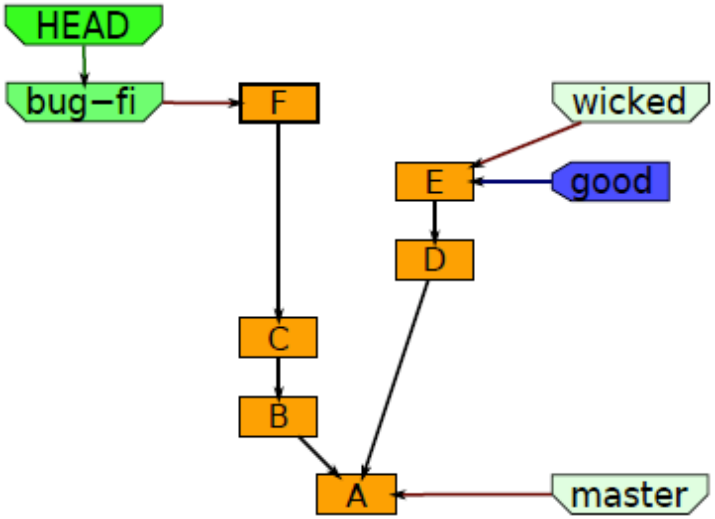
Manager asks about the bug

```
git checkout bug-fix
```



So you go back to work on it some more

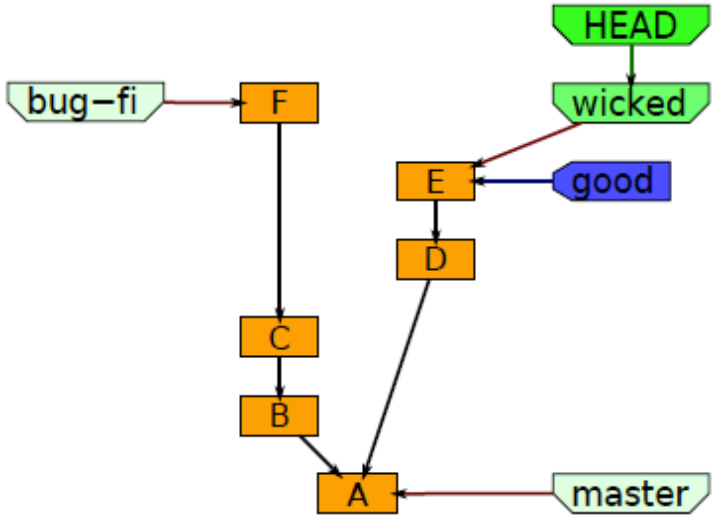
```
git commit –a –m “F”
```



http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

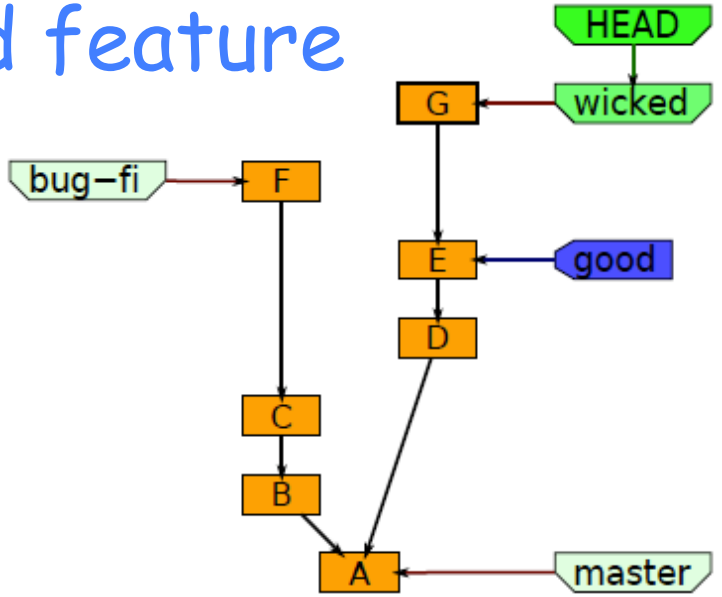
But your mind is elsewhere

```
git checkout wicked
```

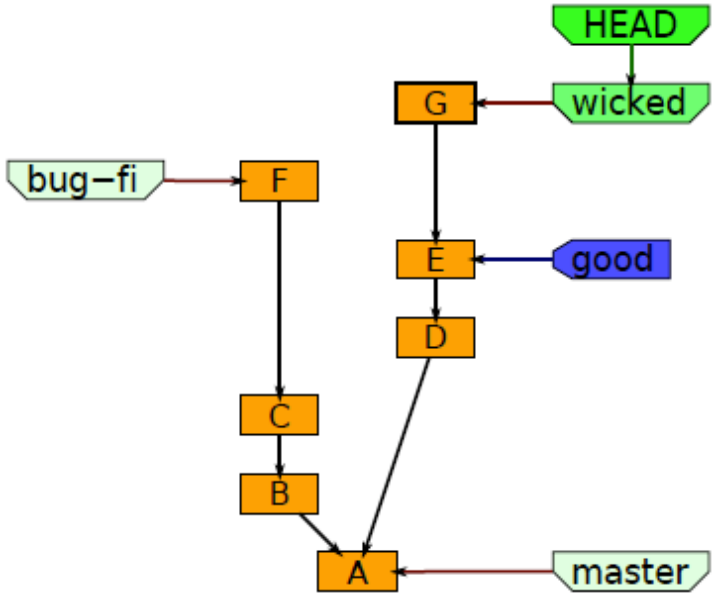


so you finish off the wicked feature

```
git commit –a –m"G"
```

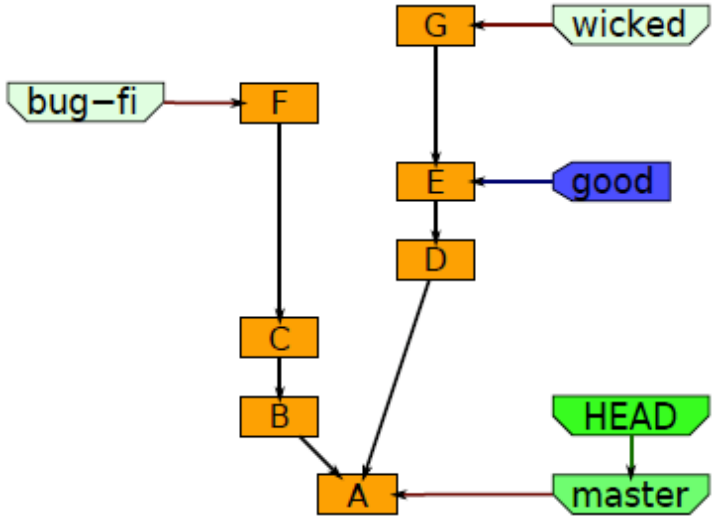


http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

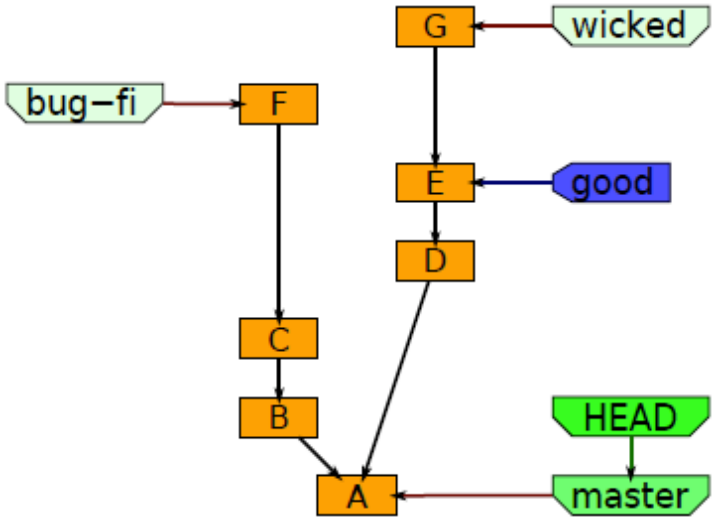


Bug fix and wicked
new feature
are both done,
so it's time to merge

```
git checkout master
```

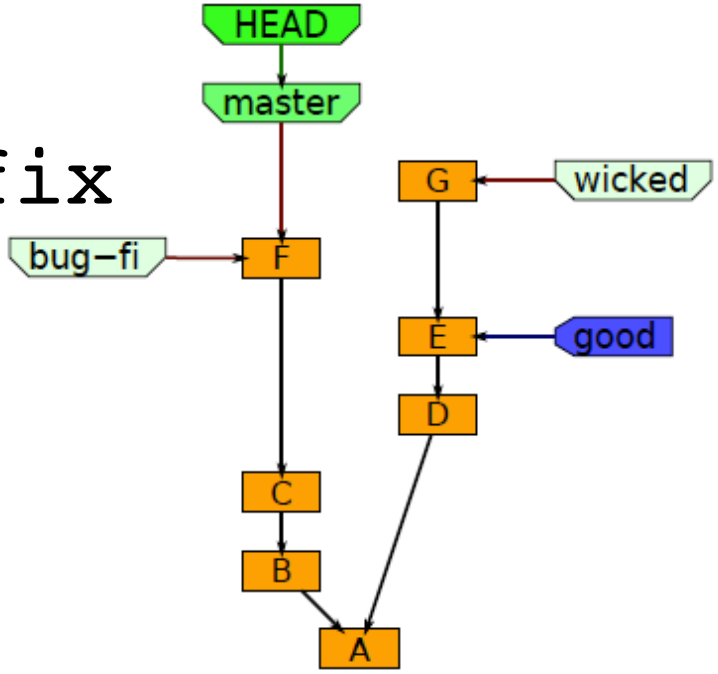


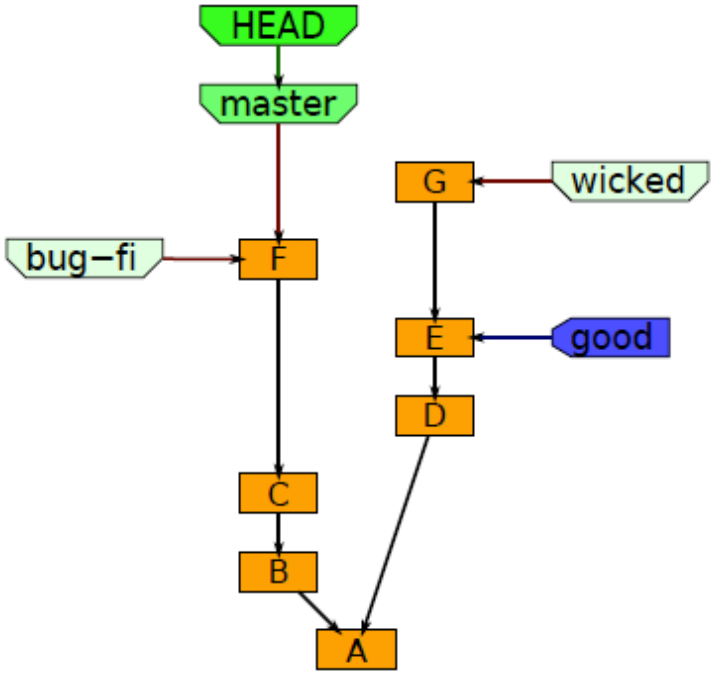
http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf



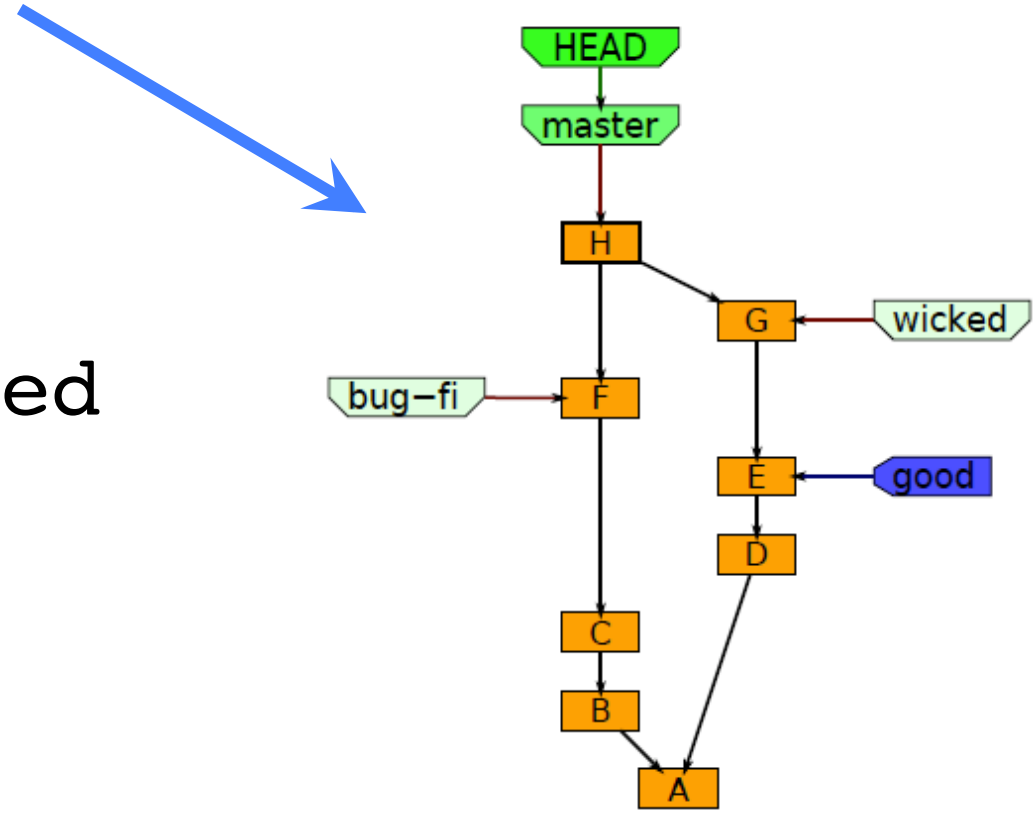
Advance the
the master to include
the bug fix

```
git reset --hard bug-fix
```





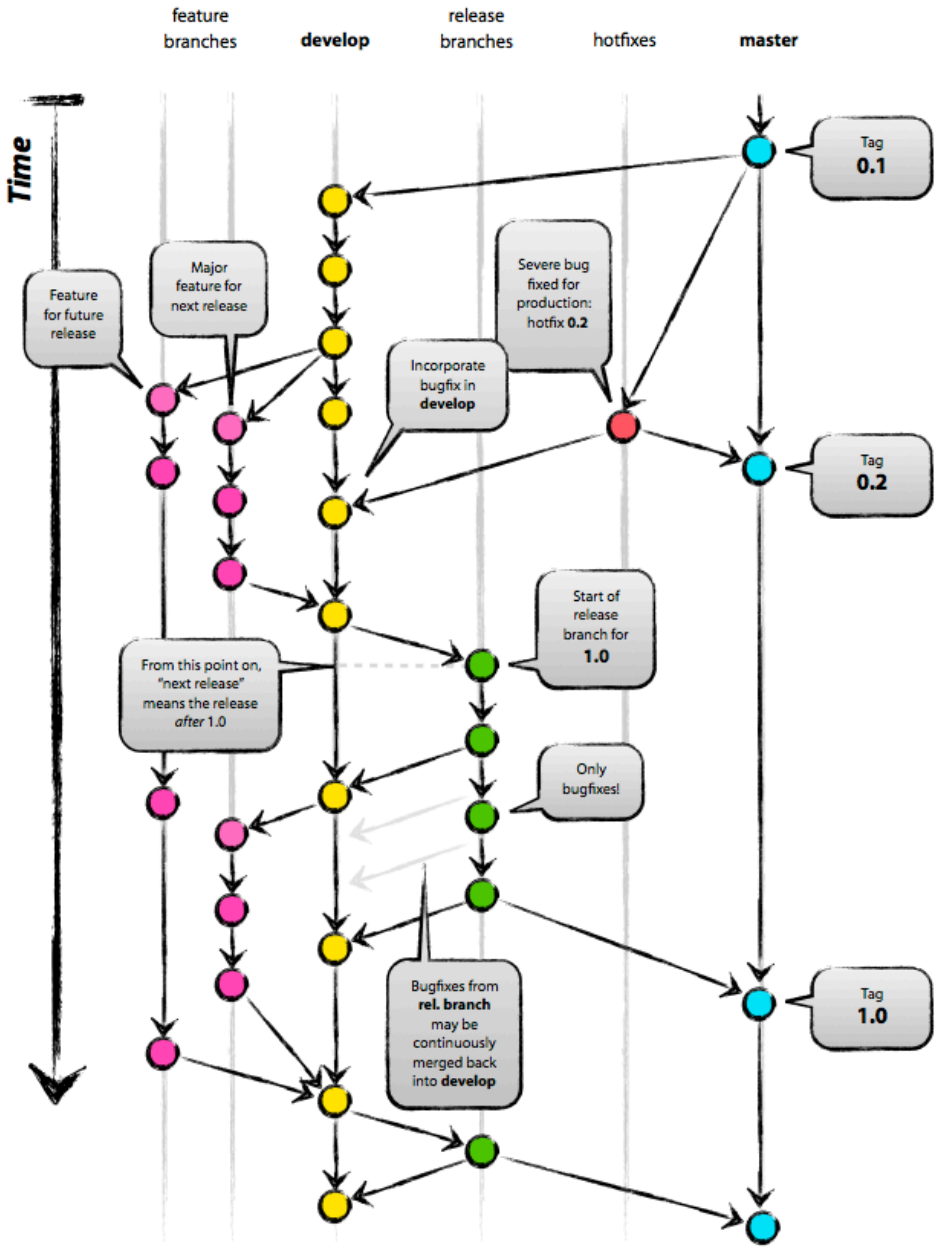
Merge it into the master branch




```
git merge wicked
```

http://edgyu.excess.org/git-tutorial/2008-07-09/intro-to-git.pdf

For a more complete description of branch management and the the commit-rebase-merge cycle, see the separate notes at the website linked below.



http://nvie.com/wp-content/uploads/2009/12/Screen-shot-2009-12-24-at-11.32.03.png