Persistent

Git –

Branching, Merging and Rebasing with Git

Persistent University

Key Learning Points

1. **Managing Branches (Rename, Delete, Recover)**
2. **Branching Workflows**
3. **Merging Branches**

Git Branching

Branching

- GIT has changed the way developers think of merging and branching, with GIT merging and branching are extremely cheap and simple and they are considered core parts of your daily workflow.
- Branches are one of the most powerful features in Git, in large part because of how easy they are to use. It's as if Git wants you to branch, and getting the most out of Git will mean using branches often and effectively

When to use Branching?

- Do you have a new Idea?
- Do you want to work on some BUG's?
- R&D?
- Not sure what you doing is correct?
- **THEN think of creating a BRANCH…!!!**

Branching…Its Simple and Different…!!!

- 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”

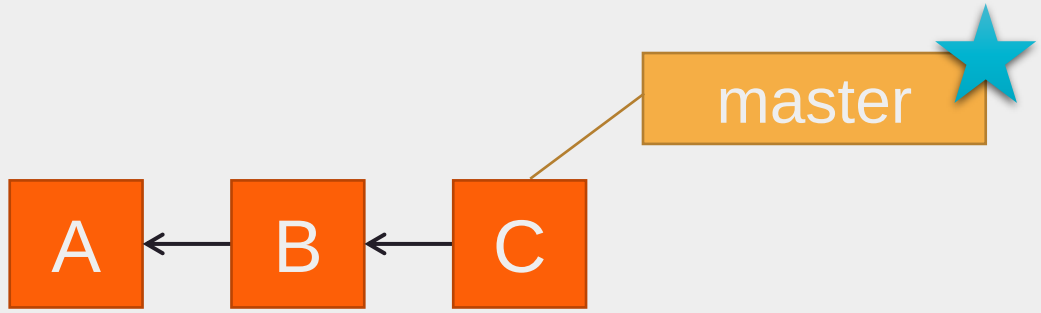
Branches Illustrated

- Assume on first commit we have ‘A’
- Assume ‘A’ is your current workspace we call it as ‘Master’ – current pointer
- ‘Master’ is the default branch created


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

Branches Illustrated

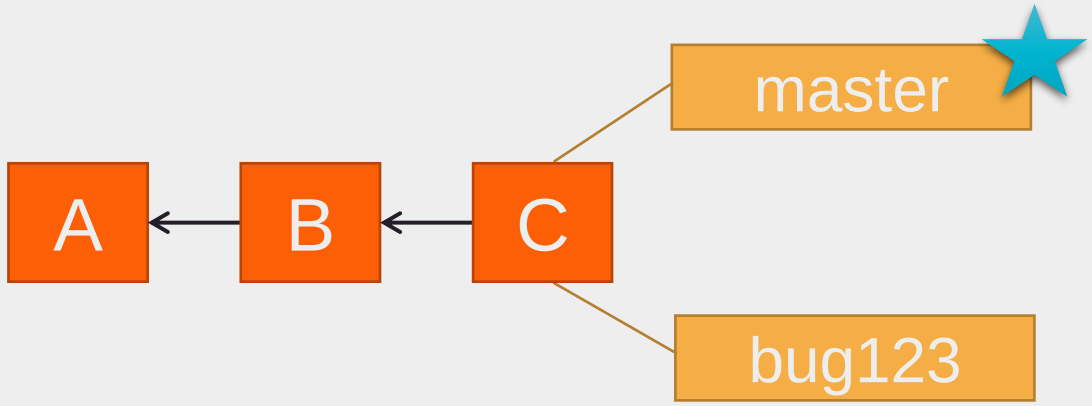
- We have made two more commits:-
- Now master is pointing to ‘C’ i.e. to last commit




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

Ohh…BUG…!!!

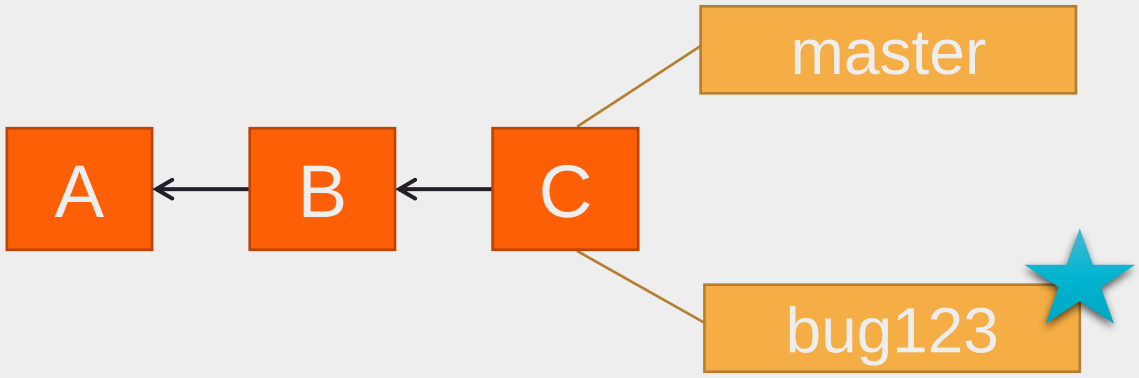
- We have a bug, then how should I work on this without disturbing master copy on which others are working.
- We start by creating local ***BRANCH*** for this bug.




```
> git branch bug123
```

Ohh…BUG…!!!

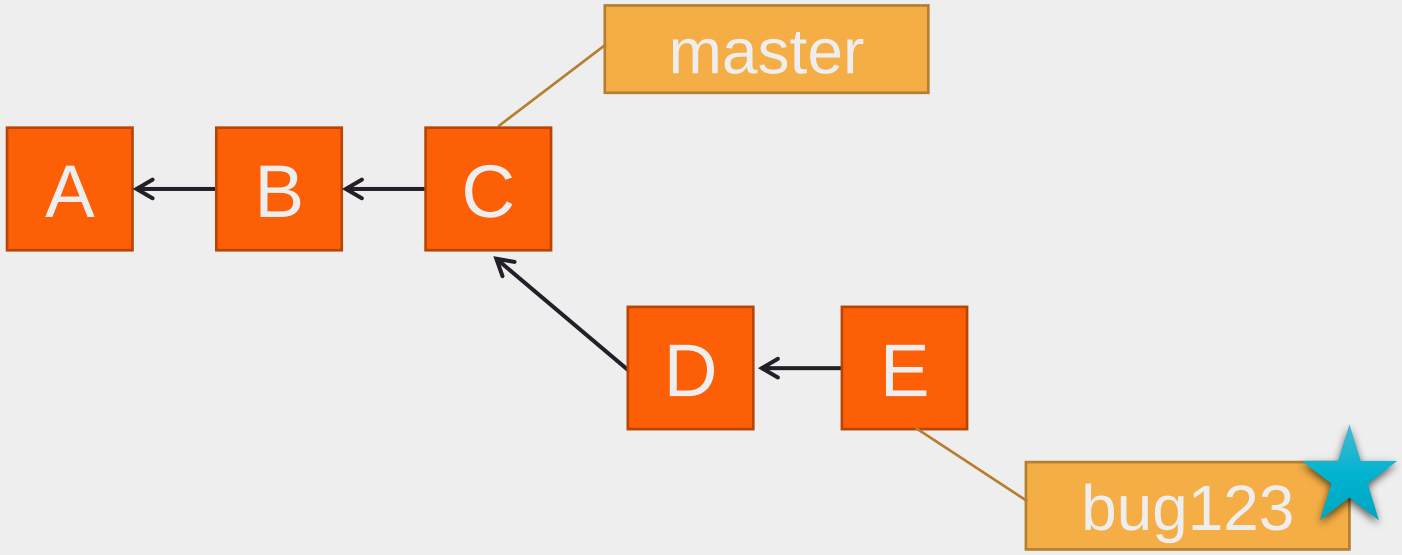
- Once branch is created we should switch to new branch in order to work it.
- Move the pointer(*) to work on the new branch




```
> git checkout –b bug123
```

Branches Illustrated

- Assume after 2 commits bug is resolved and so pointer is moving along
- Now all changes are with new branch, main master branch remains unaffected




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

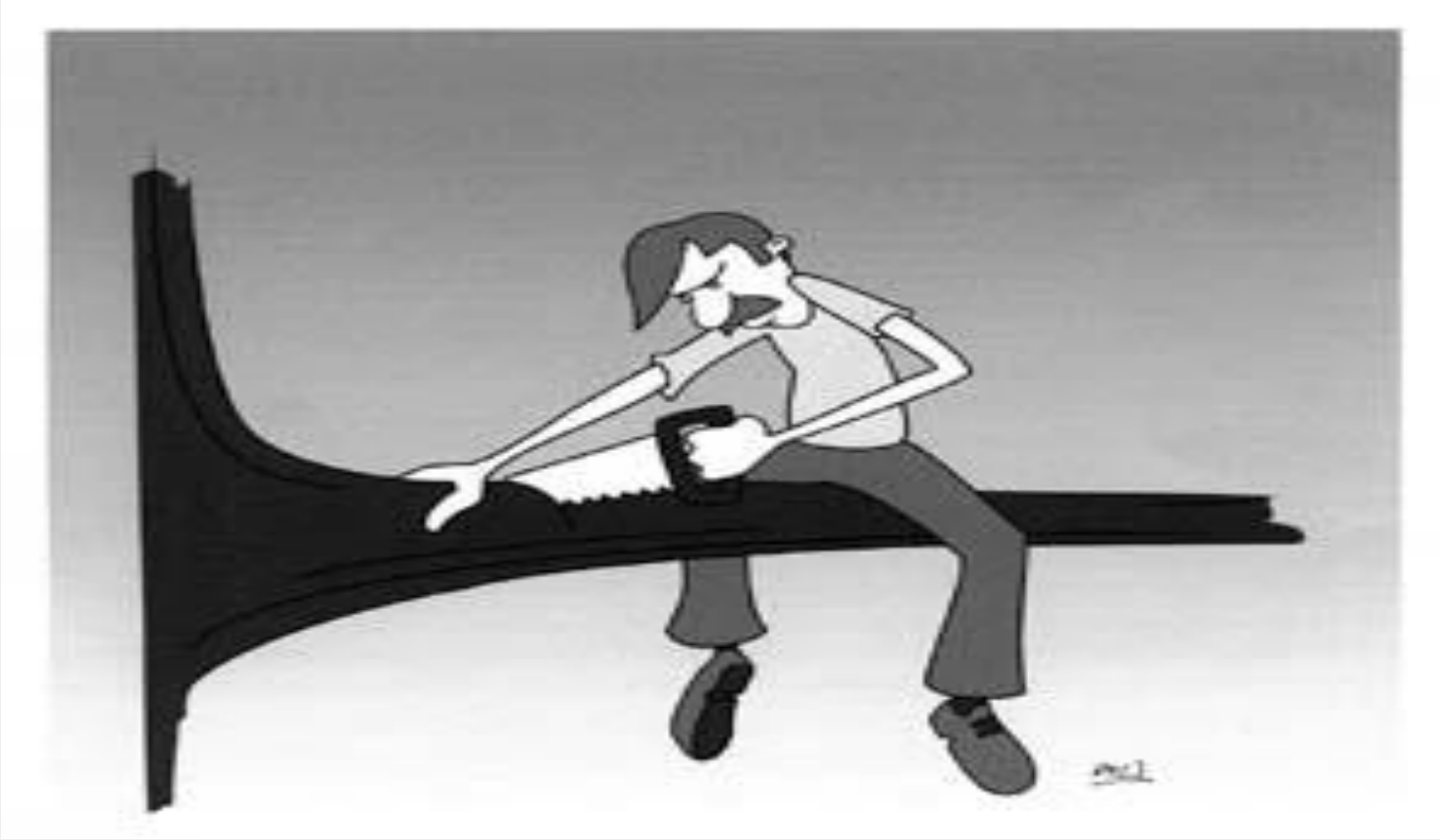
Now its time to merge.!!!

- Bug is resolved after our hard work, Now its time to take our work to main branch.
- So from where should we merge our changes?
- From current branch without checking out to main branch ???

Ohh…No, Wait…!!!

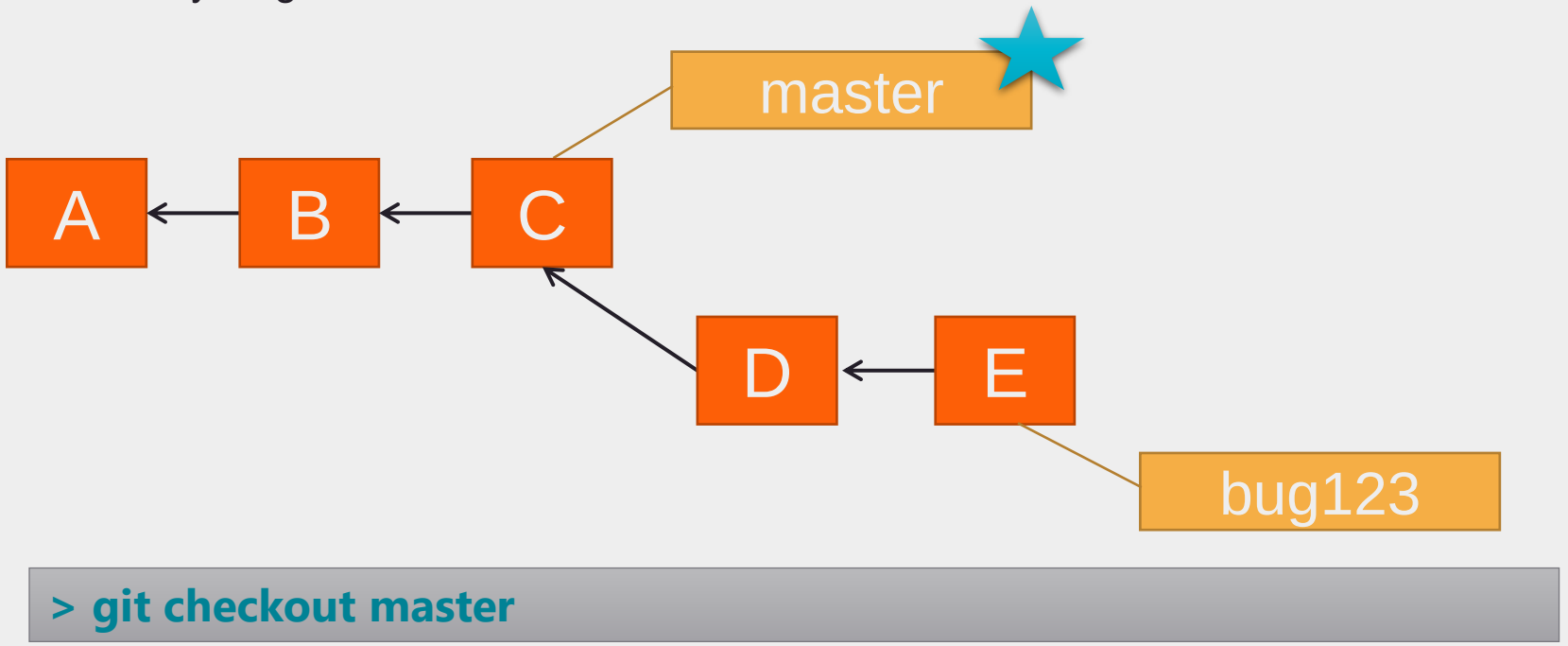
Don’t do this you are cutting out

branch on which you are sitting..



Want to see a Magic…!!! Here it is…!!!

- Before merging checkout to go back to master branch.
- *Opps…where are my changes, which I made recently.*
- Don’t worry you didn’t lose anything…

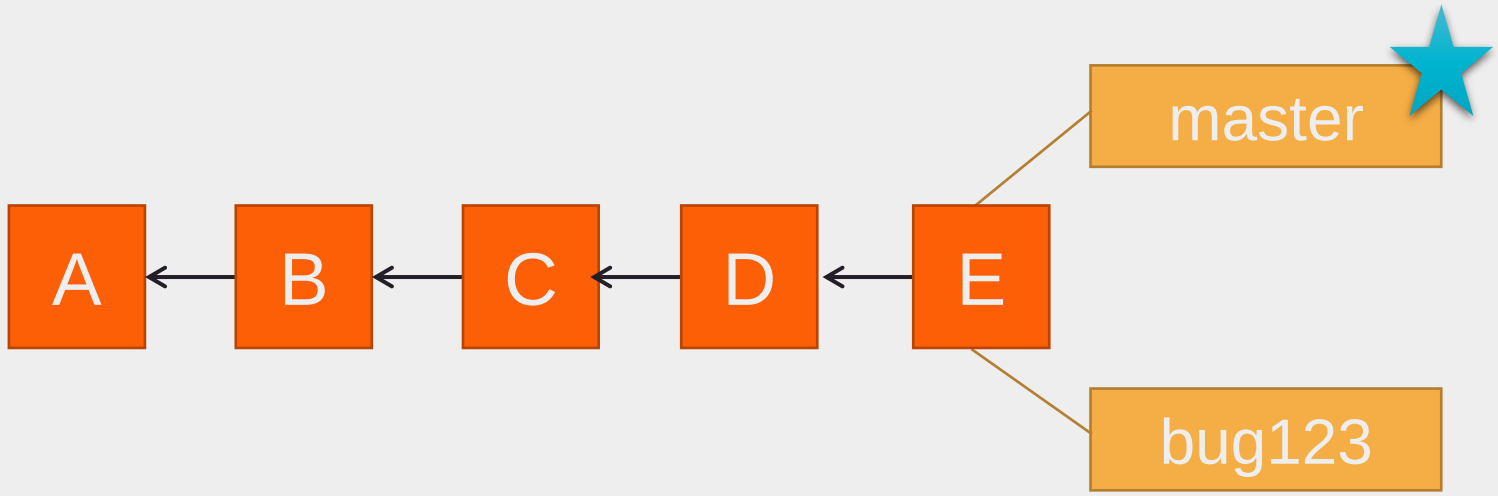



```
> git checkout master
```


Persistent

Now its time to merge.!!!

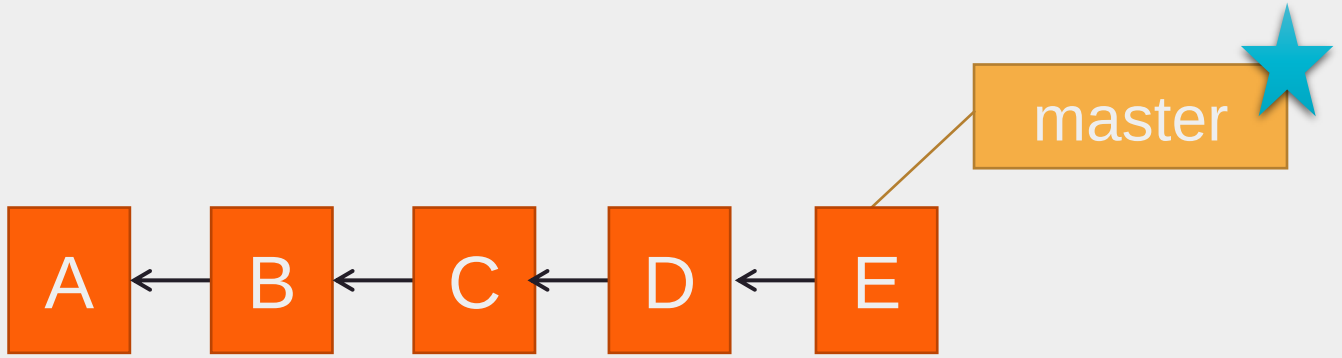
- Once we are on master branch, we can merge new changes with master branch.
- Now all our other commits made on bug123 branch will be in line with master.




```
> git merge bug123
```

Delete Branch

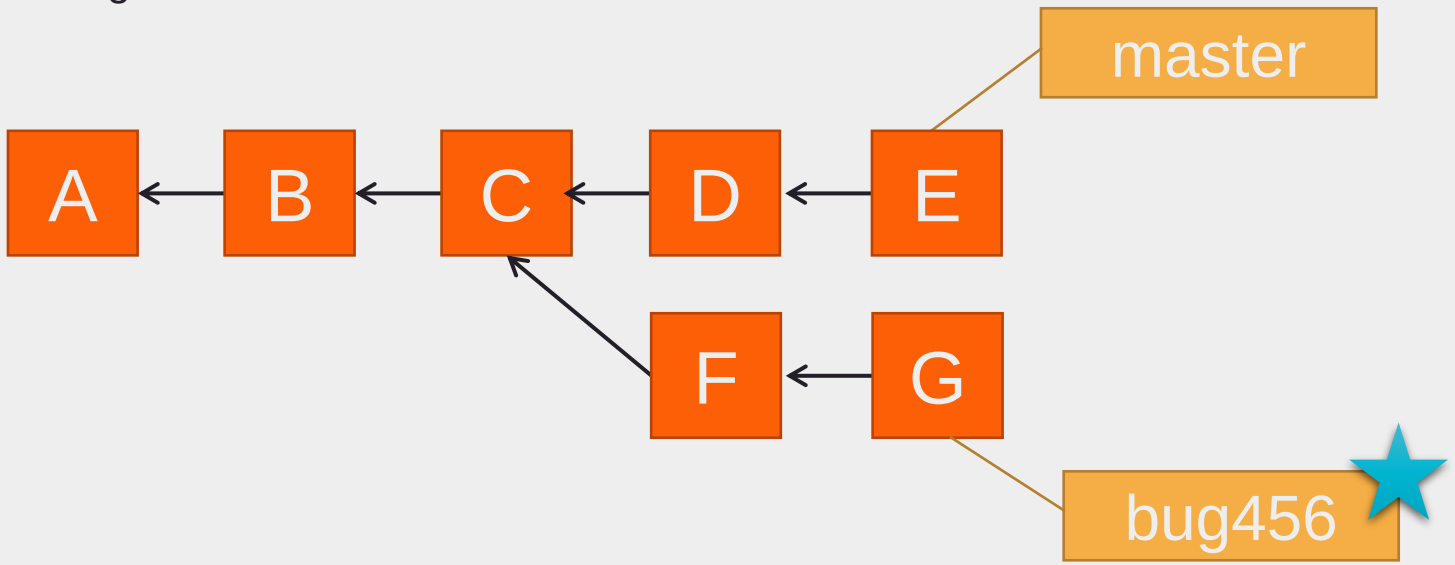
- Once we are merged all to master branch, we can delete bug123 branch.




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

Another Scenario…!!!

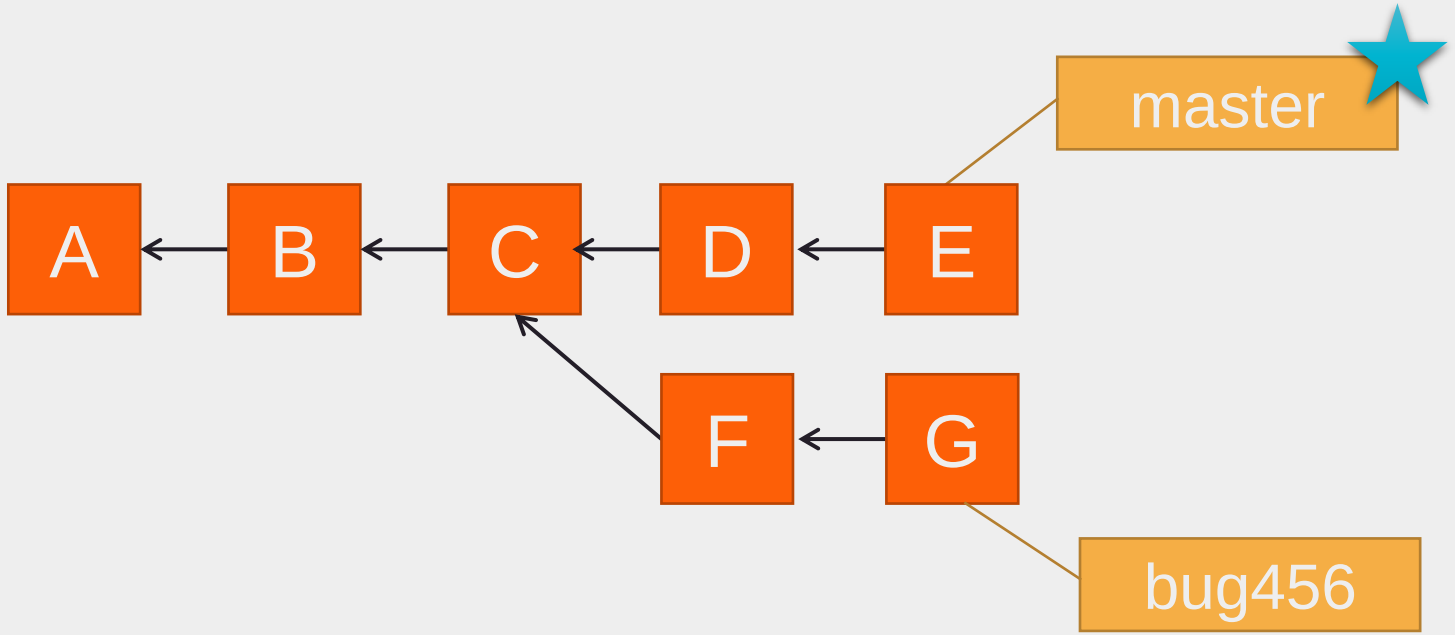
- Consider another scenario, we have bug story branch on C.
- While we were working on bug branch, others have made some changes and commits on C.
- Now we have master pointing to E not to C.



Persistent

Let’s Merge !!!

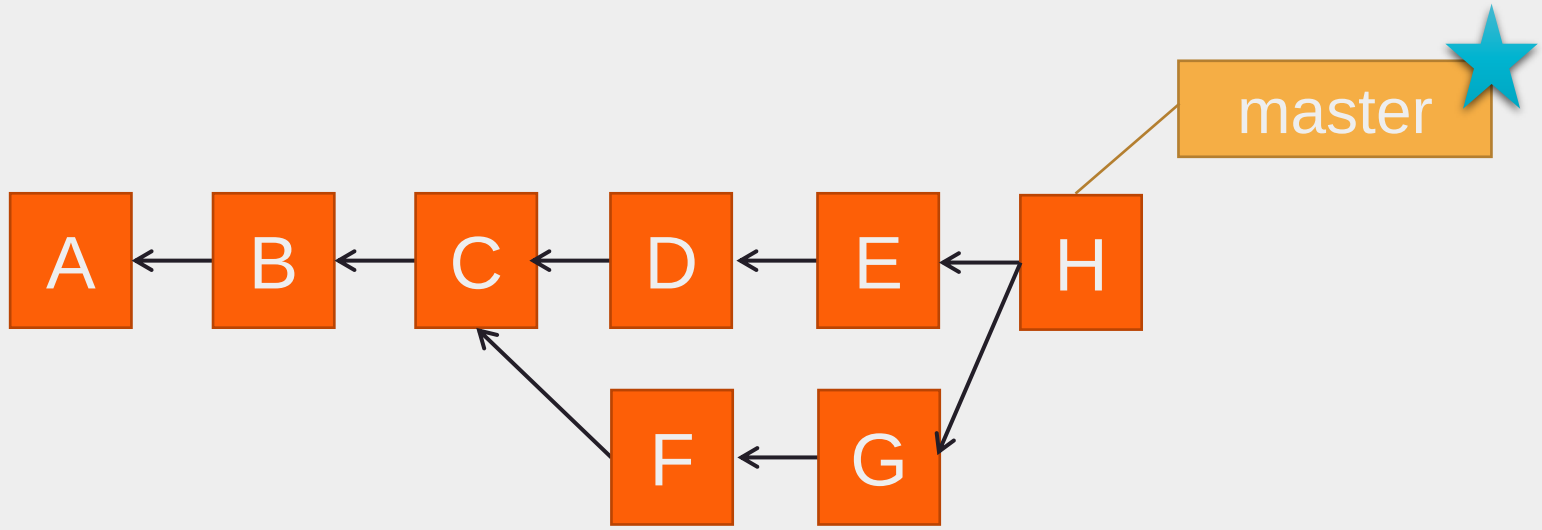
- Again to merge, we checkout back to master which moves our pointer (*).




```
> git checkout master
```

Let’s Merge !!!

- Now we merge, connecting new H to both E and G.
- Such merge can be unpleasant to perform if we have ***conflicts***

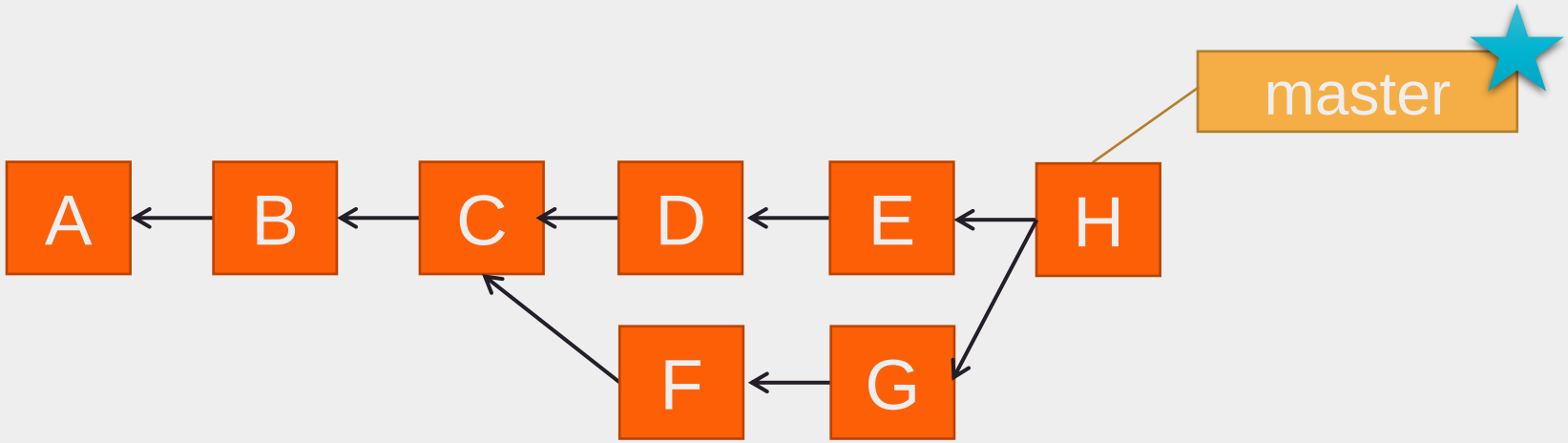



```
> git merge bug456
```


Persistent

Messy?

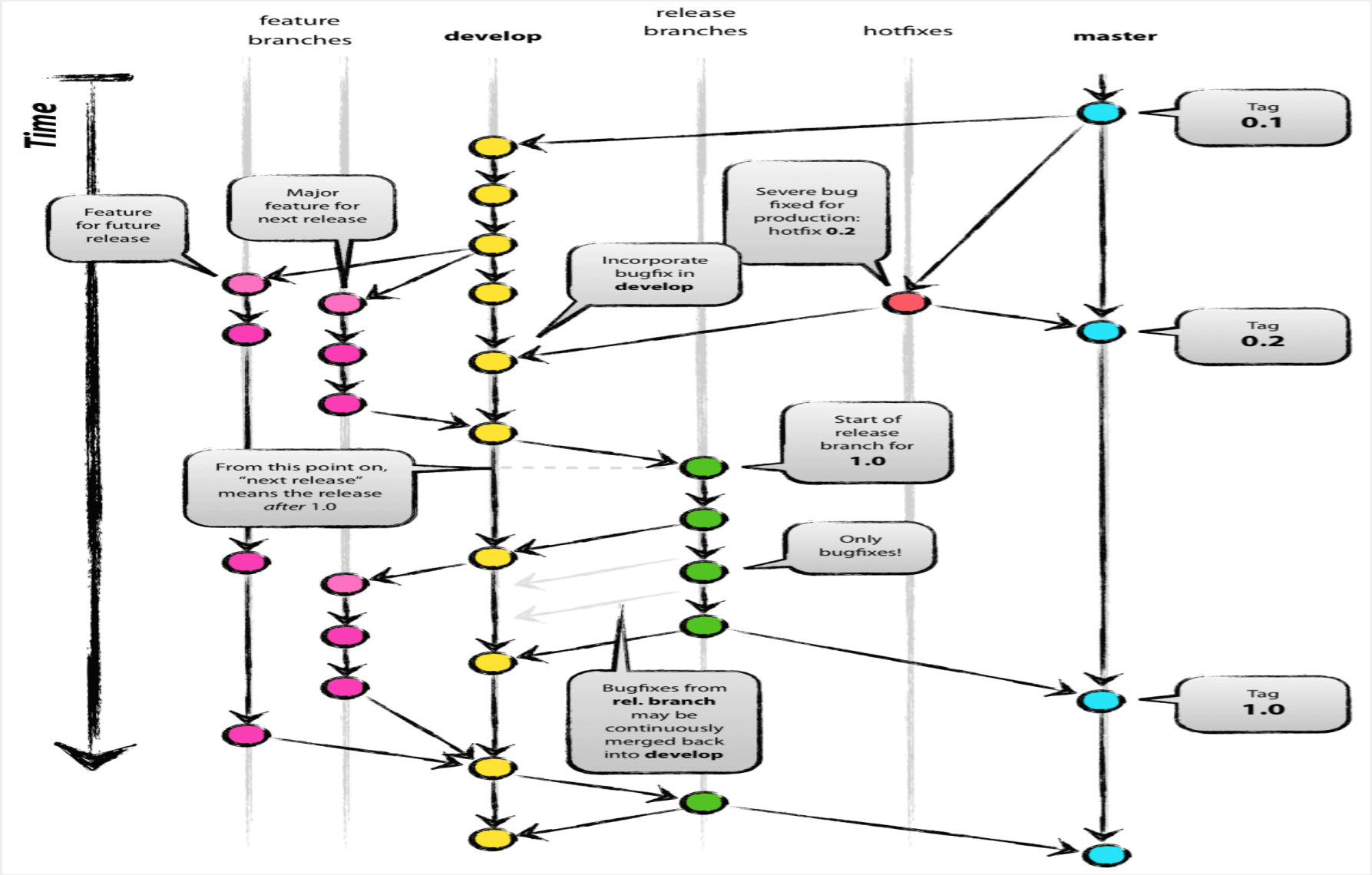
- Finally deleted the branch bug456
- Notice the structure now, this is very non-linear and can get very messy over time.




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


Persistent

Git branching workflow



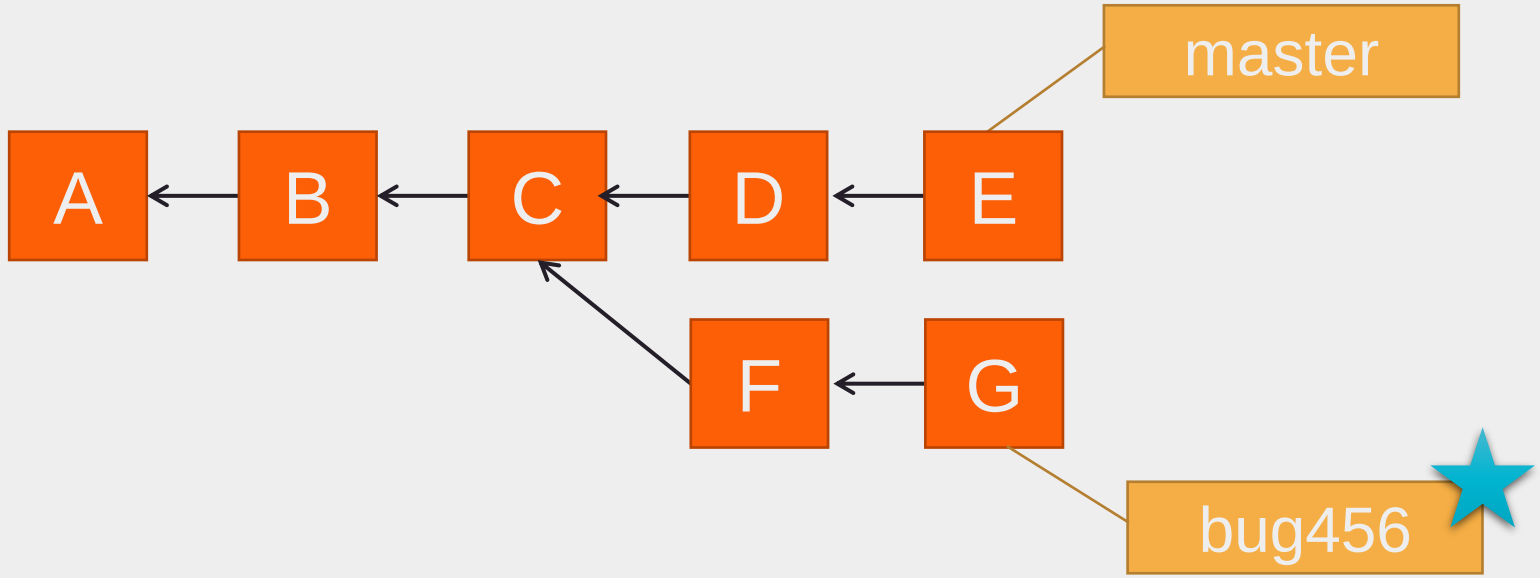
Persistent

Git Rebasing

Other Option?? Rebase

- Let’s go back in time and look at another approach that GIT enables.

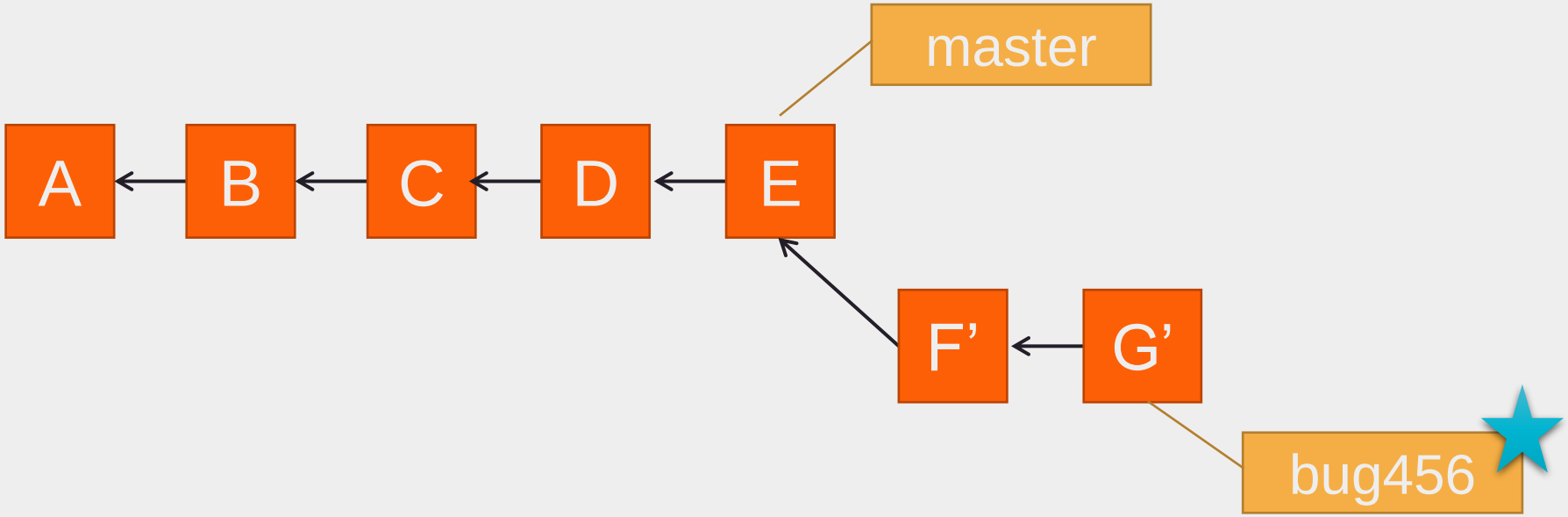
and that’s ***REBASE***



Persistent

Rebase…???

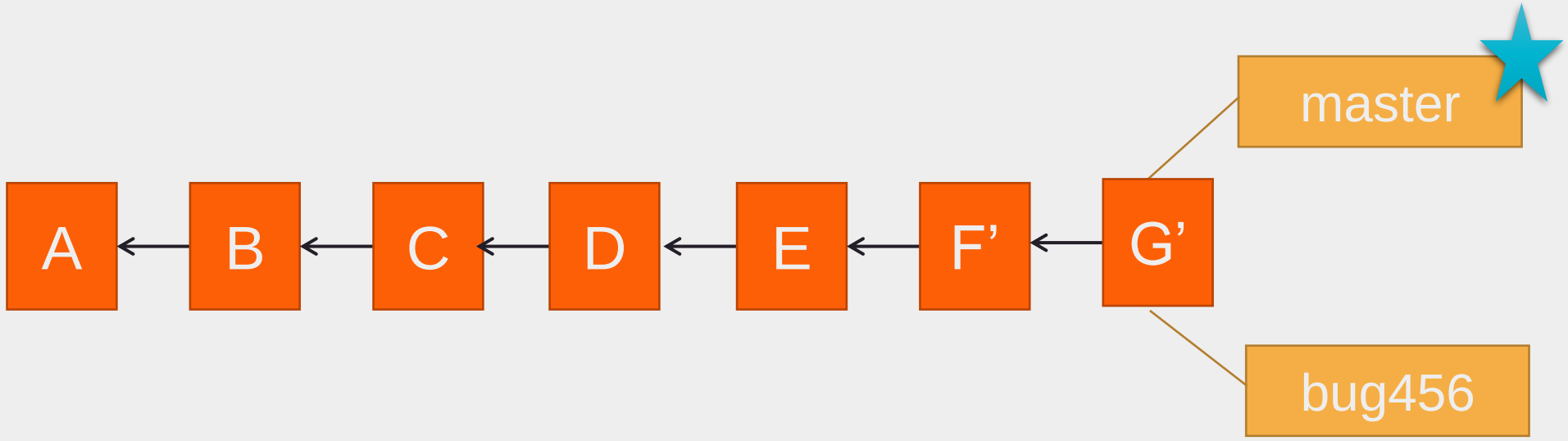
- What does this REBASE means?????
- To achieve REBASE :-
  - Take the changes made against (C) and undo them, but remember what they were
  - Now Re-apply them on E i.e. on current master.




```
> git rebase master
```

Rebase…it's Pretty.!!!

- Now this looks nice, in a Linear Flow.
- Rather than parallel, actual change set F' and G' have come after E




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


Persistent

Rename, Delete and Recover Branch

- Rename:-

```
> git branch -m old_branch new_branch
```

- Delete :-

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

- Recover :-

```
> git checkout -b <branch> <sha>
```

Conclusion

Quick and Easy to create ‘Feature’ Branches

Local branches are very powerful

Rebase is not scary

Tagging

Branch or Tag ?

- Branch is something like a separate thread
- Whereas Tag is like a Label.
- Assume you have stable release with some bug fixes say v1.0 :-
    - This version fixes bug123
    - This version fixes bug456
    - This is final v1.0

  This represents state of code.
- Use Tags to mark release points.

Tagging

- List all tags:-

  - **git tag**
  - **git tag –l “v1.3.1*”** → **tags with particular pattern**

- Creating tag with inline message:-

  - **git tag v1.0 –m "Tagging version 1.0"**

- Creating tag for longer message:-

  - **git tag –a v1.0** → **opens default editor for longer message**

Stashing

Stashing

- While resolving a bug on some branch, say we have got some high priority task
- We can't commit coz its **half done**, we can't discard otherwise, **rework**.
- Then how should we

handle this messy state

of work?

- We have ***Stash***

  to ***store temporarily***.

Stashing

- Stashing is dirty state of working directory.
- i.e. modified tracked files and staged changes – and saves it on stack of unfinished changes that we can reapply at anytime.

➢ git stash	→ saves working dir to stash
➢ git stash list	→ show all stash
➢ git stash apply	→reapply recent stash

FAQ

- What is Branching in Git?
- When to create a branch and how?
- How to switch between the branch without committing?
- How to manage branches?
- Difference between Branch and Tag?
- What is Stash and importance of Stash?

Summary

With this we have come to an end of our the session, where we discussed about

- What is Branching and Merging?
- What is Rebasing and how its different from branching?

At the end of this session, we see that you are now able to answer following questions:

- What is Branching and Merging?
- What is Rebasing ?
- What is Tagging and Staging?

In the next session we will discuss about

- Working remote repositories(GitHub)

Reference Material : Websites & Blogs

- https://git-scm.com/book/en/v1/Git-Branching
- https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging
- http://learngitbranching.js.org/
- https://www.digitalocean.com/community/tutorials/how-to-use-git-branches
- https://github.com/Kunena/Kunena-Forum/wiki/Create-a-new-branch-with-git-and-manage-branches

Reference Material : Books

- ***Pro Git***
  - By Scott Chacon and Ben Straub
  - *Publisher:* Apress

- ***Version Control with Git***
  - By Jon Loeliger, Matthew McCullough
  - Publisher: O'Reilly Media

Key contacts

Git Interactive :-

Vaishali Khatal
vaishali_khatal@persistent.com

Asif Immanad
asif_immanad@persistent.co.in

Vice President

Shubhangi Kelkar
shubhangi_kelkar@persistent.co.in

Persistent

Thank you!

Persistent University