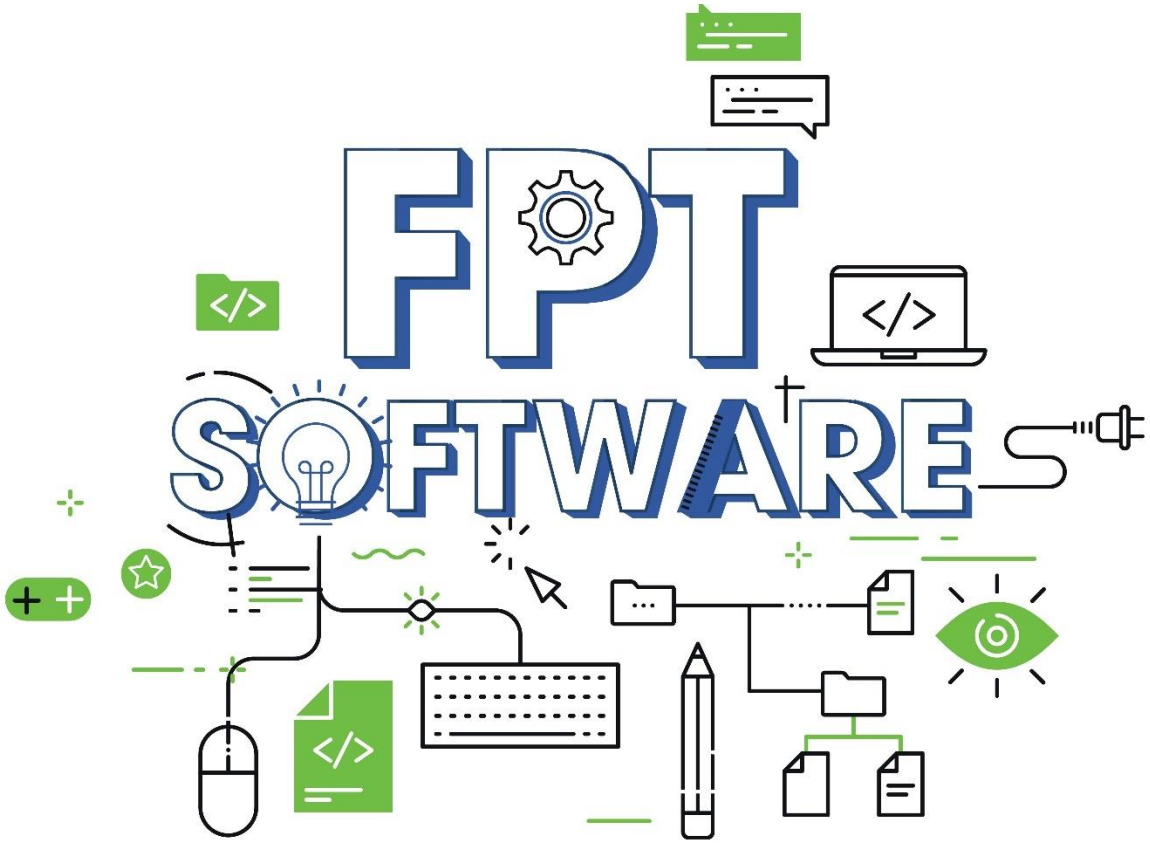
Fresher Android

Version Control with Git



Lesson Objectives

- *<Mandatory slide>*
- *<Brief the objectives of the course: What trainees learn after the course>*
- *<Remember to update the course version in Notes part>*

Section 1

Introduction to VCS Version Control System

Agenda

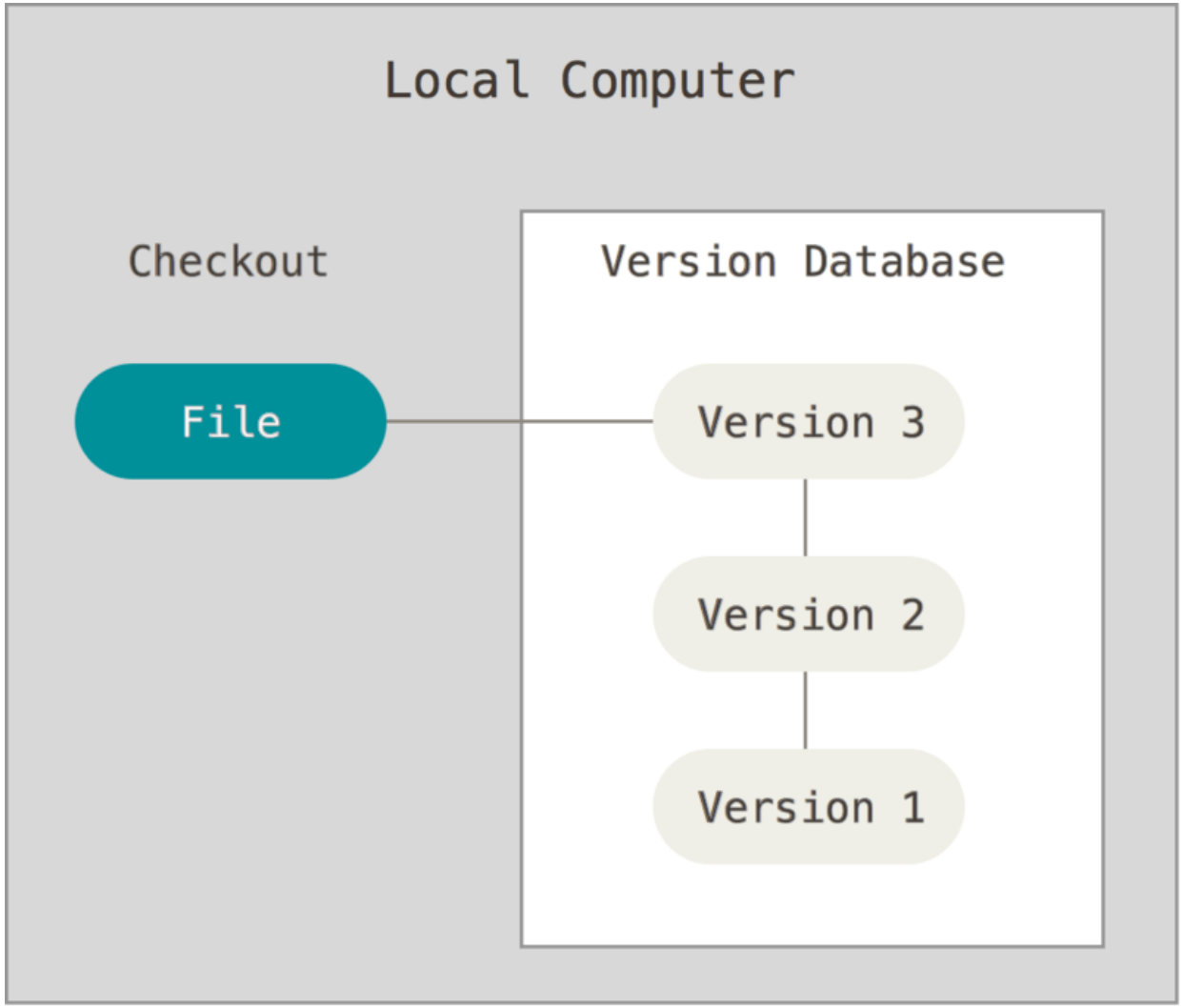
- *Types of Version Control System*
- *Brief about git*

Introduction Version Control System

- *Record changes for a repository over time, have ability to recall specific versions later*
- *Various types*
  - *Local*
  - *Center*
  - *Distributed*

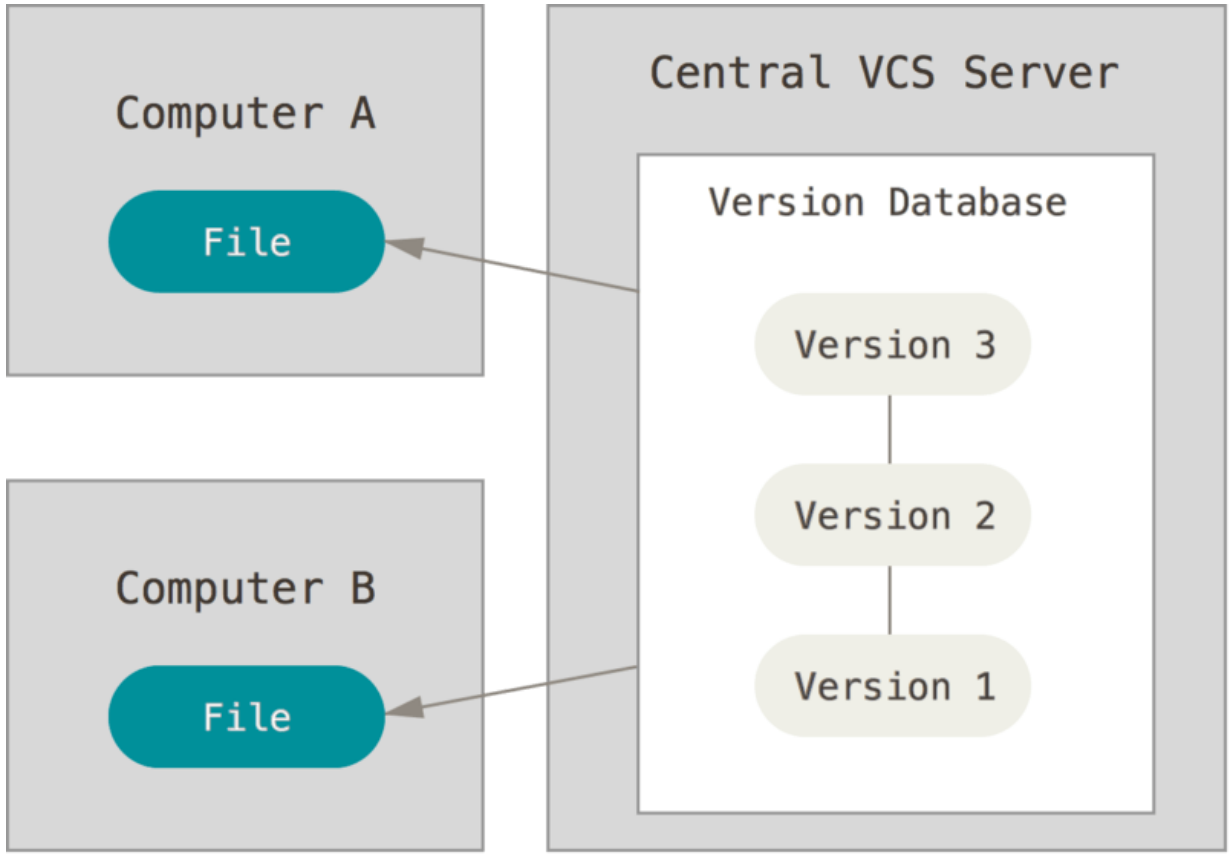
Local Version Control System

- Represent
  - GNU RCS
- Pros
  - Simple
- Cons
  - Incredibly error prone
  - Use system tiny
  - Can’t share



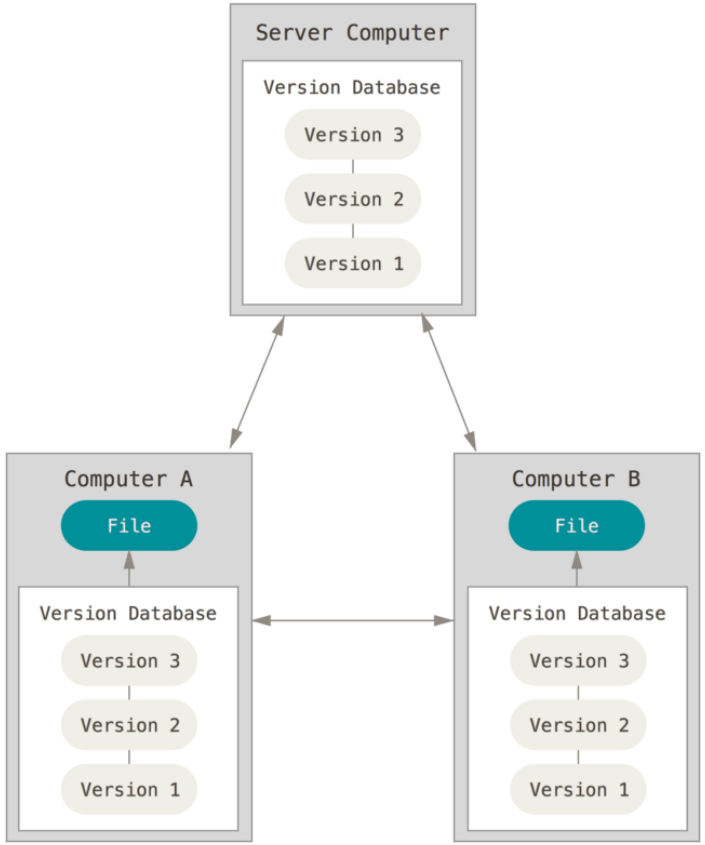
Centralized Version Control System

- Represent: CVS, Subversion, and Perforce
- A single server that contains all the versioned files, clients checkout from and commit to that central place
- Client checkout snapshot (version) of repository at a point.
- Pros
  - Everyone knows what everyone else on the project is doing at a certain degree
  - Administrators have fine-grained control overall
- Cons
  - Single point failure



Distributed Version Control System

- Represent: Git, Mercurial, …
- Have a server as a repository, clients fully mirror the repository, often support “branch”
- Pros
  - No single point failure
  - If server down, we can use another repository and mirror back to new one.
  - Developer can collaborate with different groups of people in different ways simultaneously within the same project
  - We can setup several type of workflows
  - Many operation are local: checkout, commit, sync,…



Git

- *Created by Linus Torvalds, creator of Linux, in 2005*
  - *Came out of Linux development community*
  - *Designed to do version control on Linux kernel*
  - *Init on Apr 3rd , 2005, announce on Apr 6th , PC installer released on Apr 7th , merge branch feature released on Apr 18th , used to manage Linux Kernel source v2.6.12 on June 16th*
- *Goals of Git:*
  - *Speed*
  - *Support for non-linear development (thousands of parallel branches) – Fully distributed*
  - *Able to handle large projects efficiently*

Section 2

Install Git

Install Git

- *Download site: https://git-scm.com/download*
- *Detail guide: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git*

Section 3

Basic Git

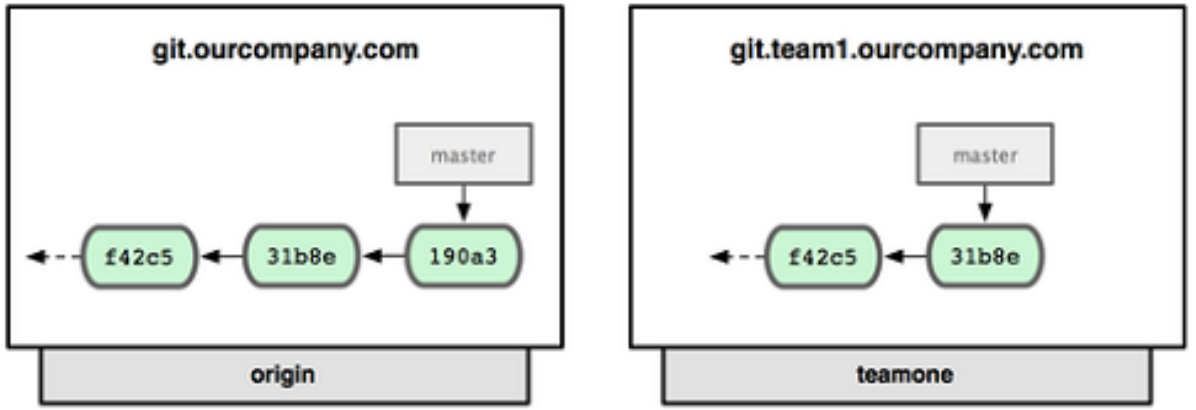
Agenda

- *Terms in Git*
- *Git Concept*

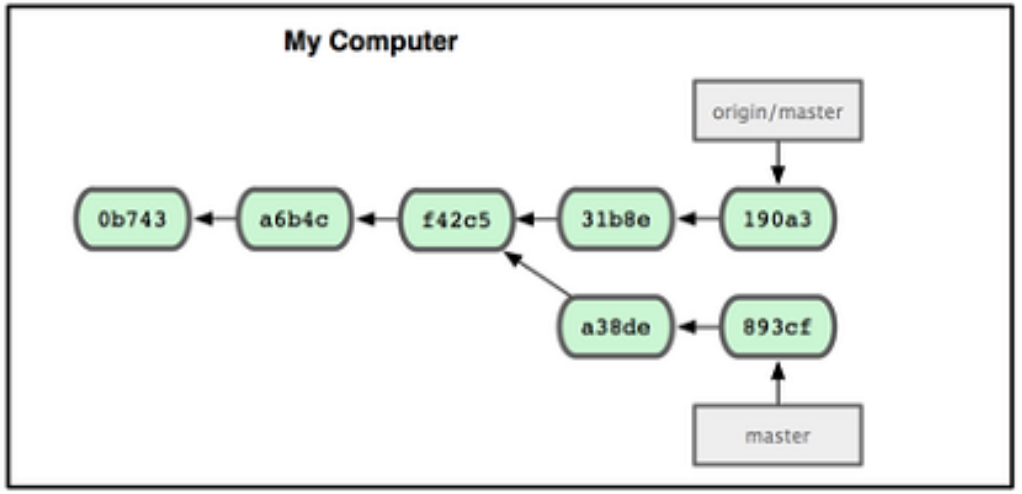
Term in Git

- Repository
  - Where document stored, grouped as working space
  - Create by `git init` then we have ***local repository***, while repositories on online server be called ***remote repository***
  - Working history stored in `.git` folder
- Remote
  - Repository on online server, default name called origin
  - All *local clones* linked with ***remotes***: 0, 1 or many

Term in Git

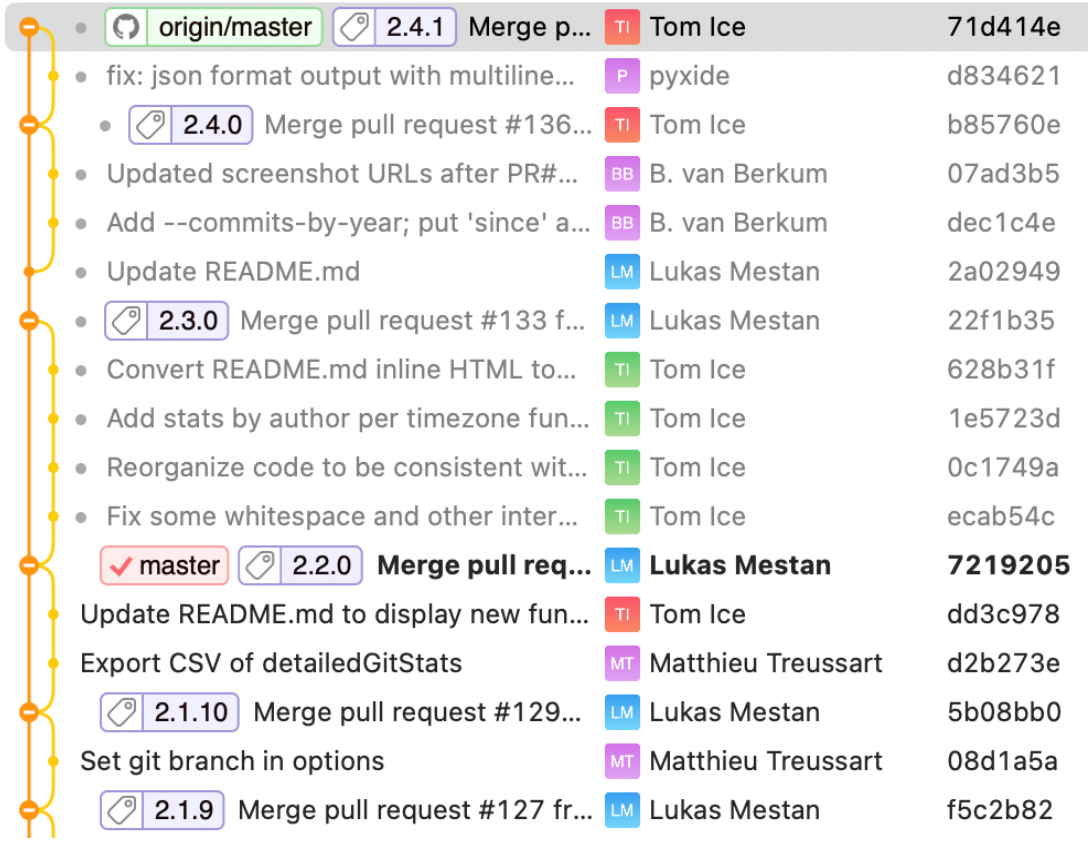



```
git remote add teamone git://git.team1.ourcompany.com
```



Term in Git

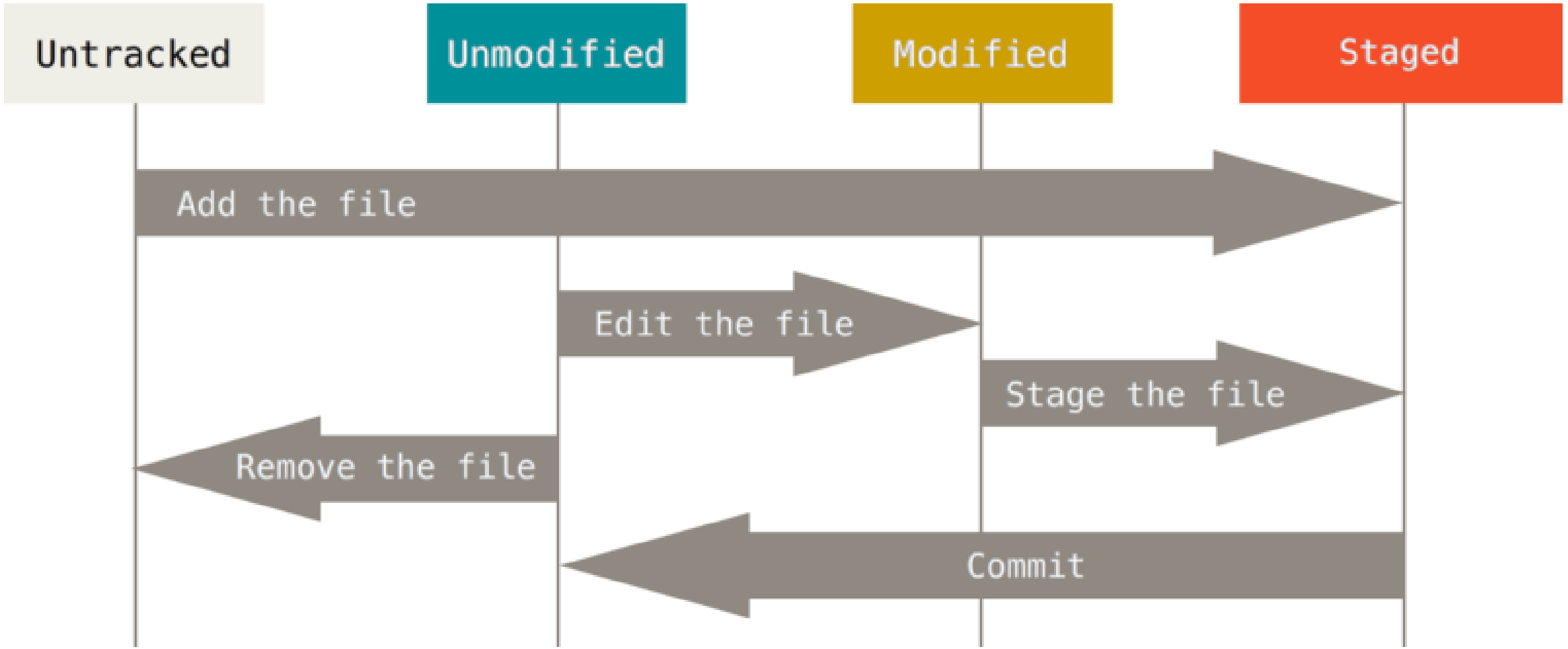
- Change:
  - Git manage the **change**, not the **snapshot**
- Commit
  - ***Commit*** is used to store the ***change***
  - ***Branch*** is a tree branch, ***Commit*** is a node or a point on branch
  - Working history or change history stored in the ***commit***
  - `git commit -m "Commit Message"` or git commit then follow the instructor with a text editor (vi, vim, nano, etc, ..)
  - Commit has a message and SHA Id



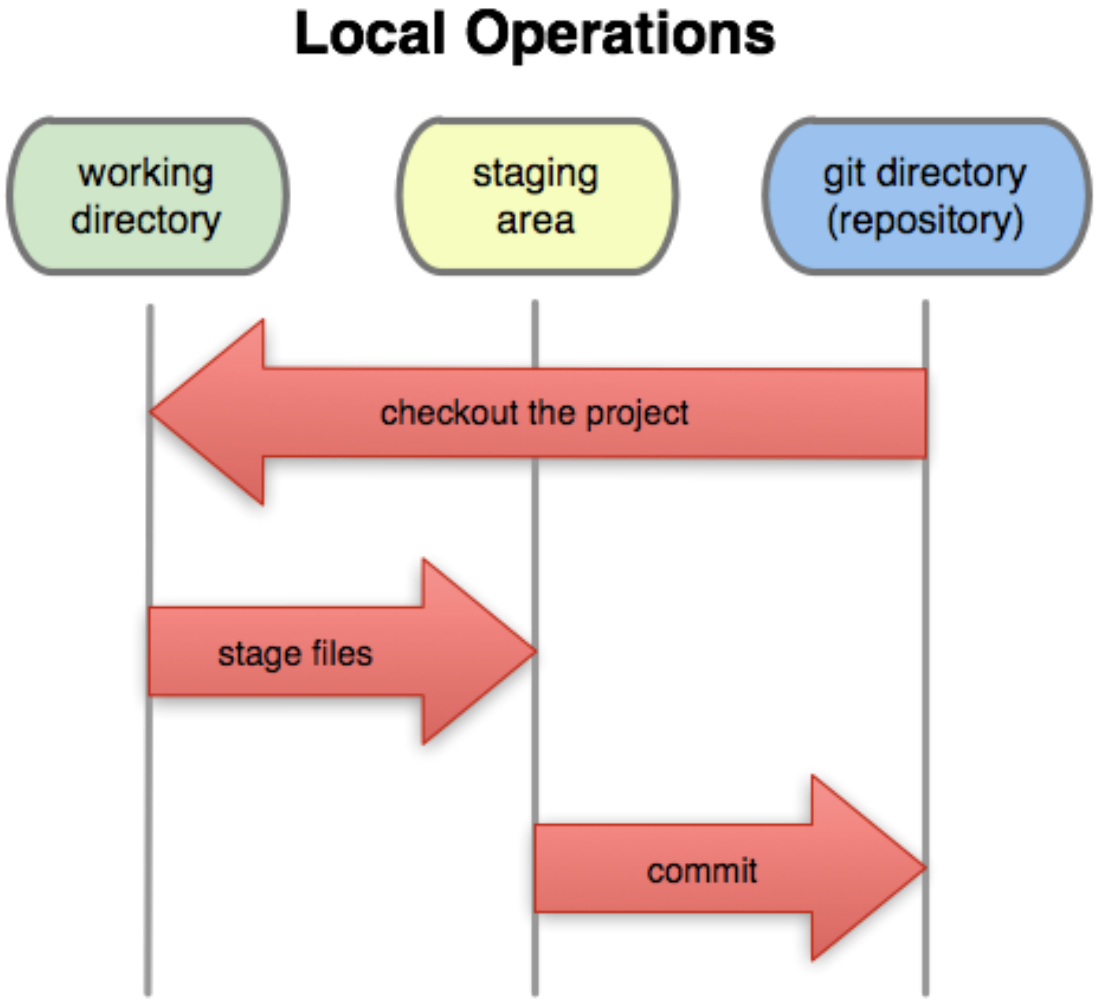
Term in Git

- Branch
  - Parallel version
  - We can work on branches without effect to ***live*** version or another's version
  - Special branch
    - main (master) – default branch
    - HEAD – the current pointer, point in commit that user is working
    - origin/master, origin/xxx, origin2/xxx are remote branch
- Checkout – change branch
  - You cloned a project/repo from GitLab and need to switch to branch develop you need: `git checkout develop`,
  - You need to add a new branch, you can add `-b`: `git checkout -b dev_another_function`

Git Concept


Untracked
Unmodified
Modified
Staged
Add the file
Edit the file
Stage the file
Remove the file
Commit

Git Concept


Local Operations
working directory
staging area
git directory (repository)
checkout the project
stage files
commit

Git Concept - Status

- **Untracked**: In working directory but git does not manage, a new file or file just removed with git rm fileName.
- **Unmodified**: Git managed file, git recognize as ***no changed*** with ***current commit*** (HEAD).
  - git add (added a new change – may be revert change)
  - git commit (save the change into commit
- **Modified**: Git managed file, git recognize there are some changes with ***current commit***
- **Staged**: ***Prepared*** to commit, we may not focus to this state. Detailed in the 2nd image.Git know about the change but the commit did not created yet. git stage.

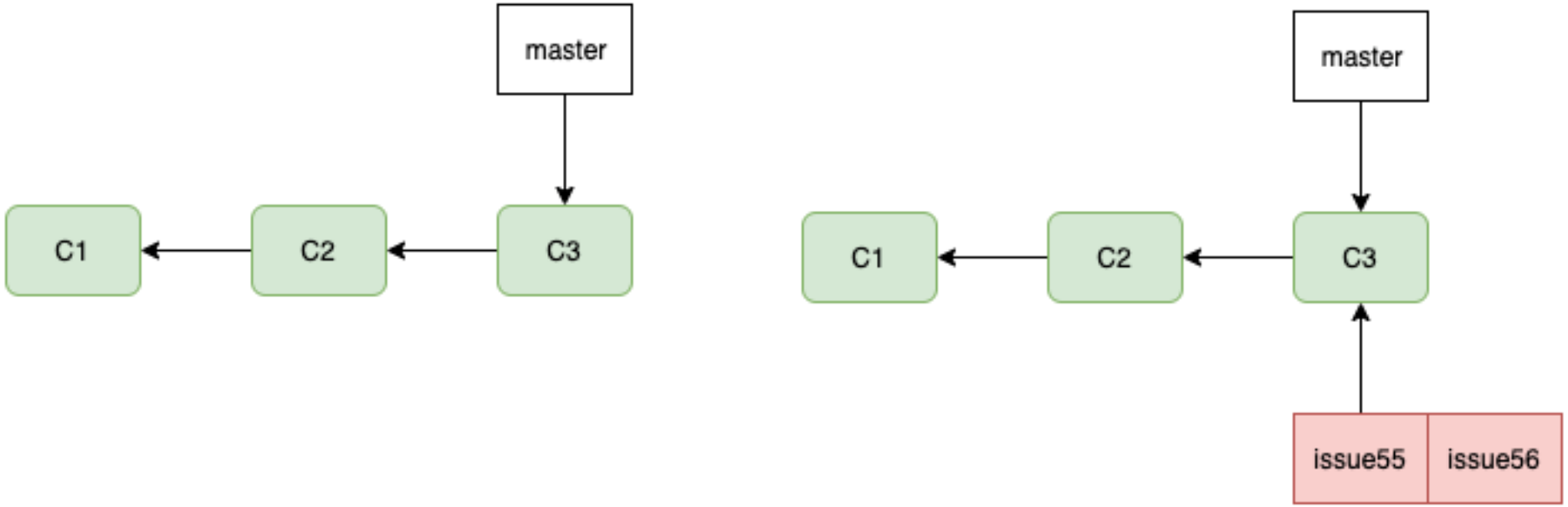
Tip with stage

- A little trick with stage, when you need to commit to **branch A** but you are working on **branch B**, you can **stage** all the files that need to be changed and **checkout** to **branch A** and **commit**. Another way most people will apply is to commit on branch B, **cherry-pick** (will talk below) commit to A, in addition, there is also a more professional way that is to create a **diff** or use **stash**

Git concept

- Merge
  - We are working in team and working in parallel,
  - We need ***combine*** works or changes together
  - We create a new commit to combine: ***merge commit***

Git merge

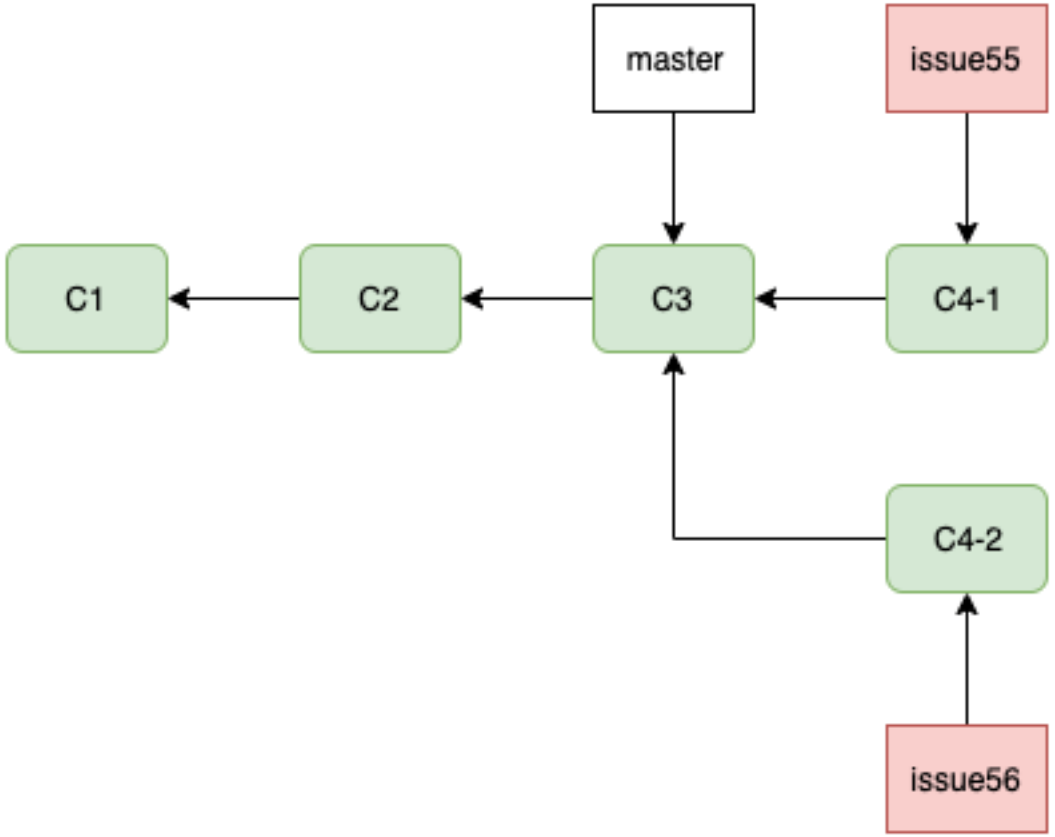


At the beginning, we have a branch: **master**

We have 2 issues 55,56. And it is often too complexity to fix, one by one

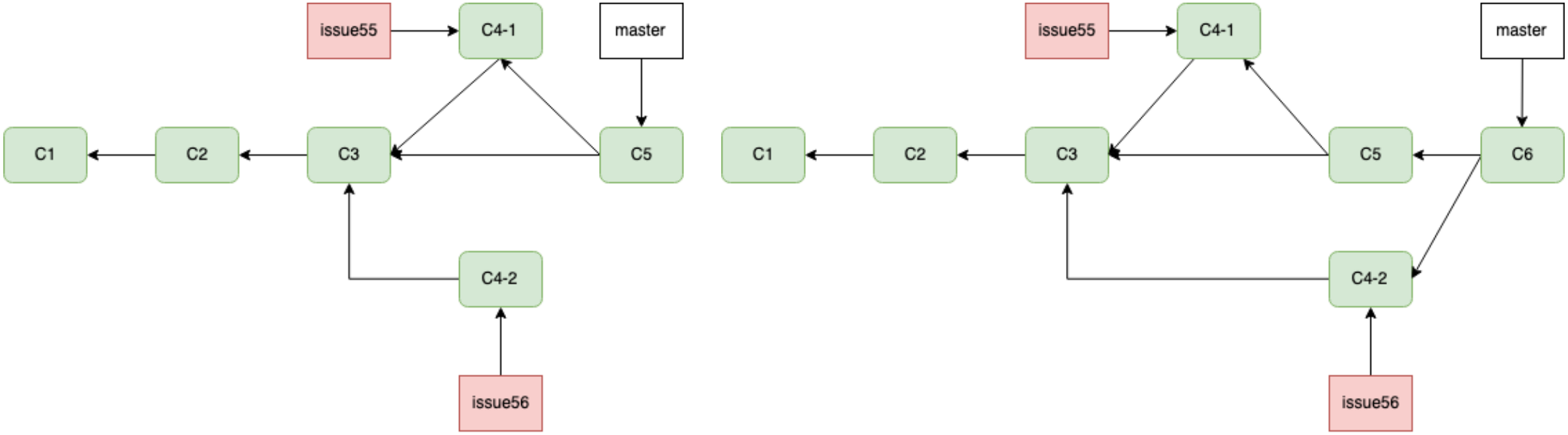
We need to create 2 branch to fix it

Git merge



Fixed it each branch the commit

Git merge




```
git checkout master # change currrent branch to master

git merge issue55

 ...

git merge issue56
```

Git merge: conflict

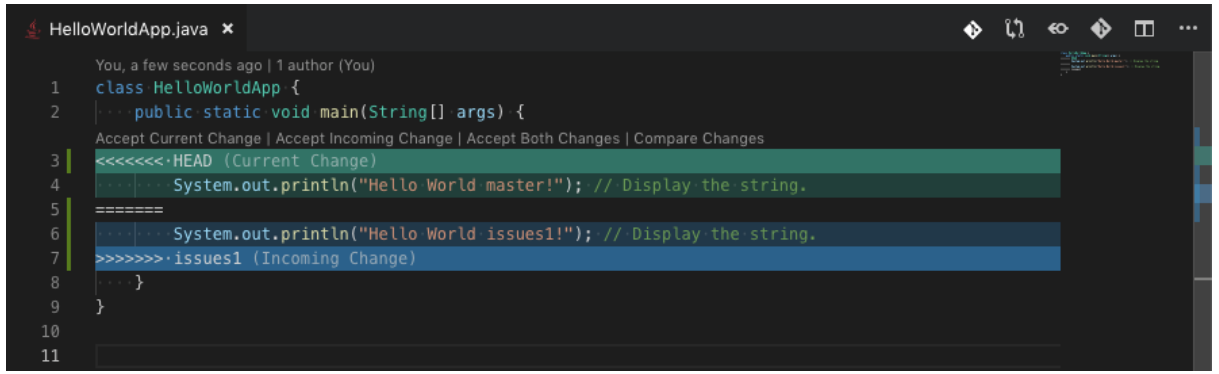
```
 mk@hades  ~/git/gitsample  master  git merge issues1
Auto-merging HelloWorldApp.java
CONFLICT (content): Merge conflict in HelloWorldApp.java
Automatic merge failed; fix conflicts and then commit the result.
```

Git merge: conflict

```
commit e8a1171ae27e3514253c5cd5c1477188da84e5b2 (HEAD -> master)
Author: Hoang Thanh Son <wingadium1@gmail.com>
Date:   Sat Apr 20 14:11:03 2019 +0700

    HelloWorld master

diff --git a/HelloWorldApp.java b/HelloWorldApp.java
index 15002bf..0423568 100644
--- a/HelloWorldApp.java
+++ b/HelloWorldApp.java
@@ -1,6 +1,6 @@
 class HelloWorldApp {
     public static void main(String[] args) {
-        System.out.println("Hello World!"); // Display the string.
+        System.out.println("Hello World master!"); // Display the string.
     }
 }
```

```
commit 985524504b38784f25a398df075382fb730092f6 (issues1)
Author: Hoang Thanh Son <wingadium1@gmail.com>
Date:   Sat Apr 20 14:09:19 2019 +0700

    HelloWorld issuse1

diff --git a/HelloWorldApp.java b/HelloWorldApp.java
index 15002bf..7d88c55 100644
--- a/HelloWorldApp.java
+++ b/HelloWorldApp.java
@@ -1,6 +1,6 @@
 class HelloWorldApp {
     public static void main(String[] args) {
-        System.out.println("Hello World!"); // Display the string.
+        System.out.println("Hello World issues1!"); // Display the string.
     }
 }
```

```
commit 1c9bba27143644dba0584745265460f268cd38
Author: Hoang Thanh Son <wingadium1@gmail.com>
Date:   Sat Apr 20 14:08:16 2019 +0700

    Init commit with HelloWorld

diff --git a/HelloWorldApp.java b/HelloWorldApp.java
new file mode 100644
index 0000000..15002bf
--- /dev/null
+++ b/HelloWorldApp.java
@@ -0,0 +1,6 @@
:
```



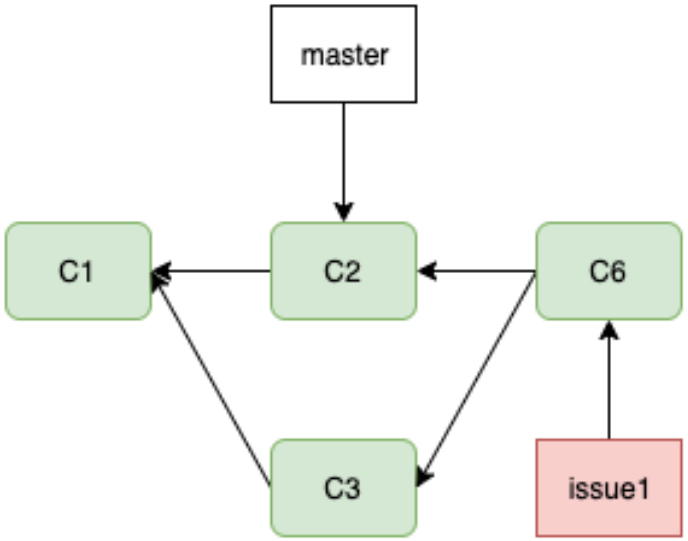
Git merge conflict

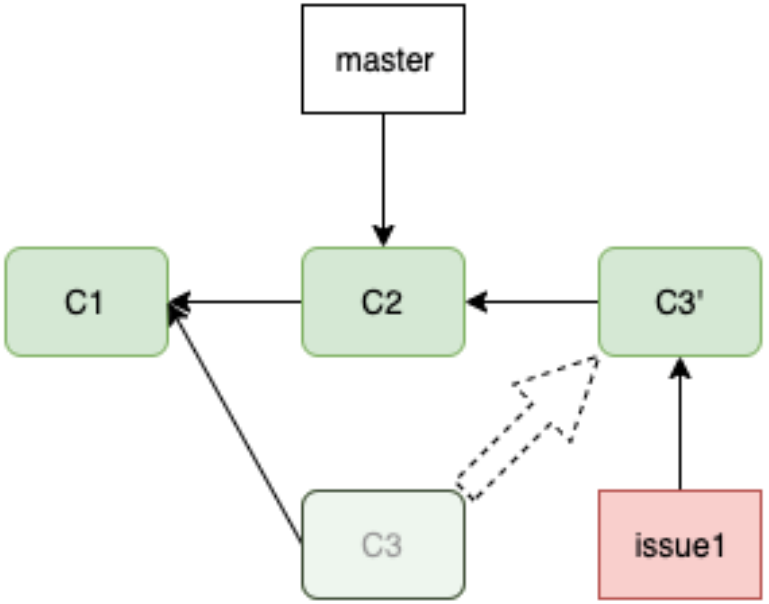
- Remember: Never ***resolve a conflict*** when you don't ***know*** what the other person just changed, really.

Git rebase

- Same with merge, rebase used to apply the changes from this branch to another branch
- We did not create a merge commit, we ***reapply*** the changes (one by one) to ***destination branch***

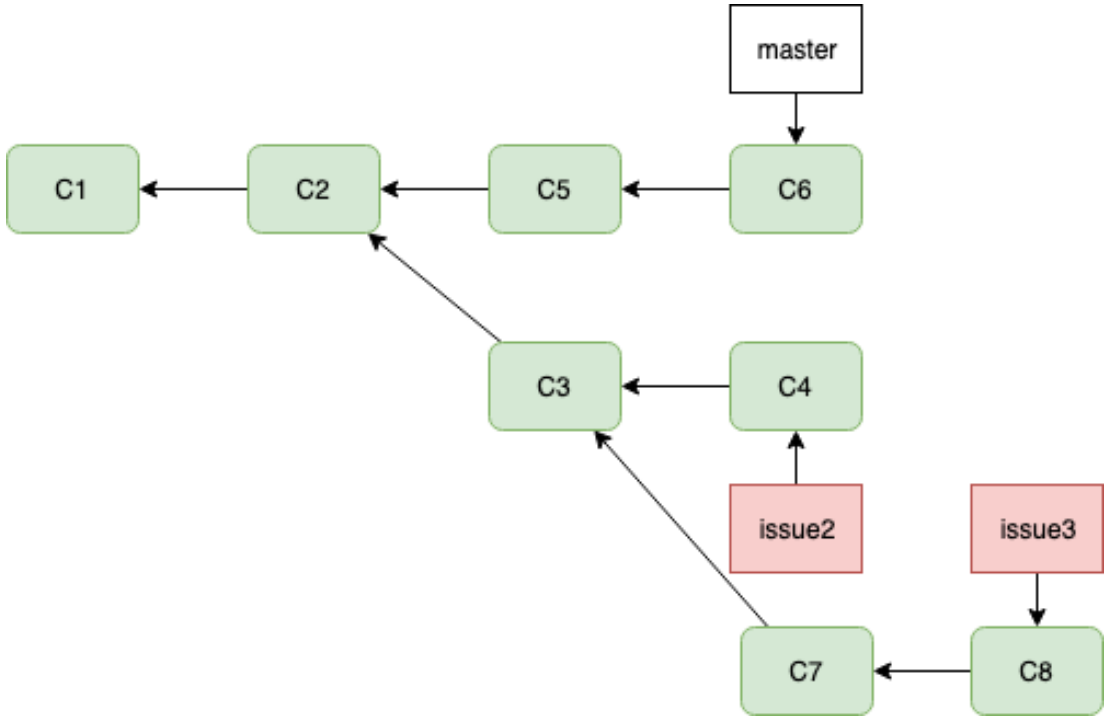
Git rebase

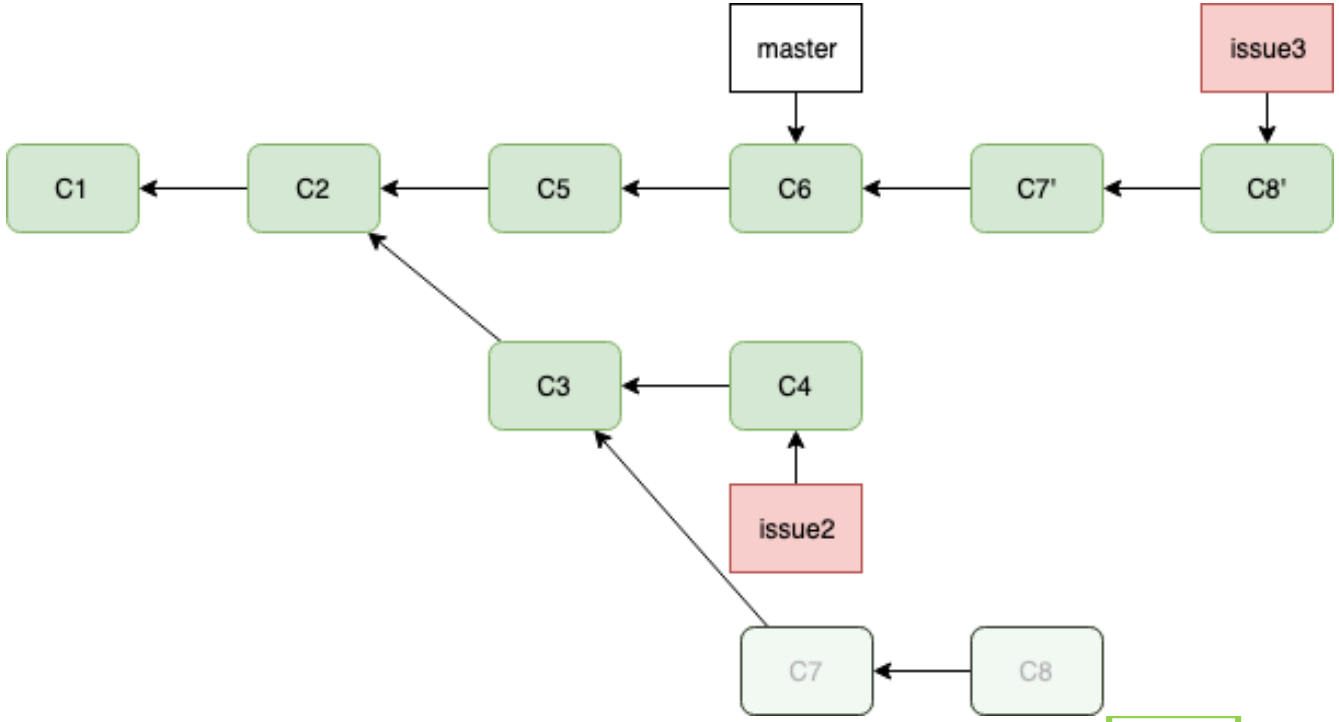





```
git checkout issues1
git rebase master
.... # conflict and resolve
git add HelloWorldApp.java # add conflict file to tracking
again git rebase --continue
```

Git rebase advance





Git rebase

- ***Attention***: do not ***rebase*** commits that you have ***pushed*** to the ***repository***. Otherwise everyone will hate and everyone will humiliate and despise.
- Basically, when you rebase you drop the commits and move them somewhere else. When you do some rebase, from a branch you pull from the repository, and push up. People will have to re-apply their commits and there will be a lot of conflict when you pull change their commits to local.

Git command

command	description
`git clone url [dir]`	copy a Git repository so you can add to it
`git add file`	adds file contents to the staging area
`git commit`	records a snapshot of the staging area
`git status`	view the status of your files in the working directory and staging area
`git diff`	shows diff of what is staged and what is modified but unstaged
`git help [command]`	get help info about a particular command
`git pull`	fetch from a remote repo and try to merge into the current branch
`git push`	push your new branches and data to a remote repository
others: `init, reset, branch, checkout, merge, log, tag`	

Working with remote

- git push
- git pull/fetch
- git remote add/remove
- git protocol
  - git clone file:///opt/git/project.git / git remote add local_proj /opt/git/project.git
  - git clone ssh://user@server:project.git
  - git clone git://user@server:project.git
  - git clone http://example.com/project.git

Summary

- Understand about VCS
- Git and git terminologies
- Git command

THANK YOU!