Version Control Systems

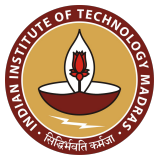
Dr. Sridhar Iyer, IIT Bombay
Dr. Prajish Prasad, FLAME University
Ankur Parmar, IIT Madras

Some images taken from http://git-scm.com/book/en/

Problems that we want to solve

- Software developers work together in large teams. How to handle concurrent changes to a project or file?
- How to manage several versions of the software? Is keeping multiple copies of source code a way to go?
- How to track changes, and also who did those changes?

So we need some way to do all this efficiently.



Scenario 1 - Working in team

Several developers are working on a Project.

One developer changed some code that caused a failure and now the system cannot start. Either

1. Identify and fix the issue. Could take a long time to debug as you don't know the change history, till then no other team member can test their new code changes as the bug will not let the system to start.
2. Or just roll back to the previous version of code that was running stable.

If you chose to rollback, **how to rollback easily?**

Scenario 2 - Maintaining different revisions/versions

Consider a company with hundreds of clients of a software system. Every client specific feature needs a new version to be released and maintained.

Huge code base, code duplicated.

- Several versions result in multiple copies of code.
- Difficult to maintain and administer these copies manually.

We need a tool that does this for us without creating complete copies of the code base.

Version Control Systems

- A system that helps in tracking and managing changes to the **source code** or other documents, and maintaining versions of the code.

- In Software Engineering, version control systems falls under software configuration management.

Feature

VCS supports

- Tracking and managing changes to the source code over time.
  - Through Branching, Merging, Commits etc.
- Maintaining different revisions of the source code (with some labelling). E.g. 1.2, 1.3, 2.0, 3.2.1
- Maintaining change history that includes tracking who changed what.

Additional features

(not an exhaustive list)

- Comparing different revisions.
- Have tools integrated for reviewing the code changes.
- Tools to track issues in your software.
- Providing ways for collaborative development in teams.

Why are they important?

- Increases productivity.
- Better communication and collaboration.
- Saving space on multiple revisions through diffs/snapshots.
- Better efficiency even as teams scale.

Nowadays the question is not weather to use it or not

But

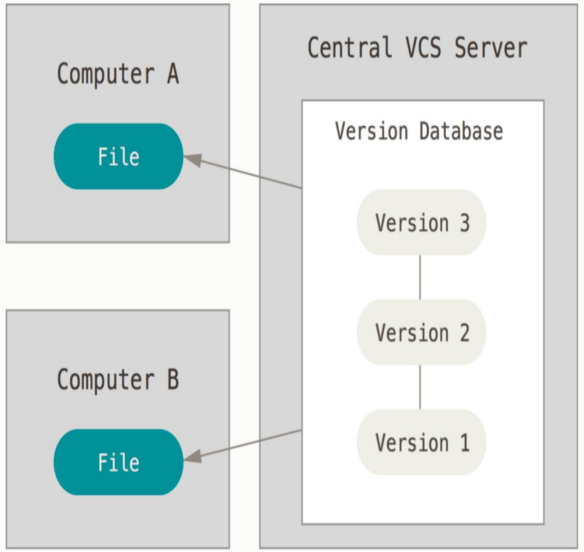
Which one to use?

Types of VCS - Centralised

- Centralised repository maintained on the server, that everybody shares.
- Entirely based on Client-server model.
- Checkout whole or part of the repository

  -> Make changes

  -> Push back the changes to the server.
- E.g. Perforce, SVN etc.

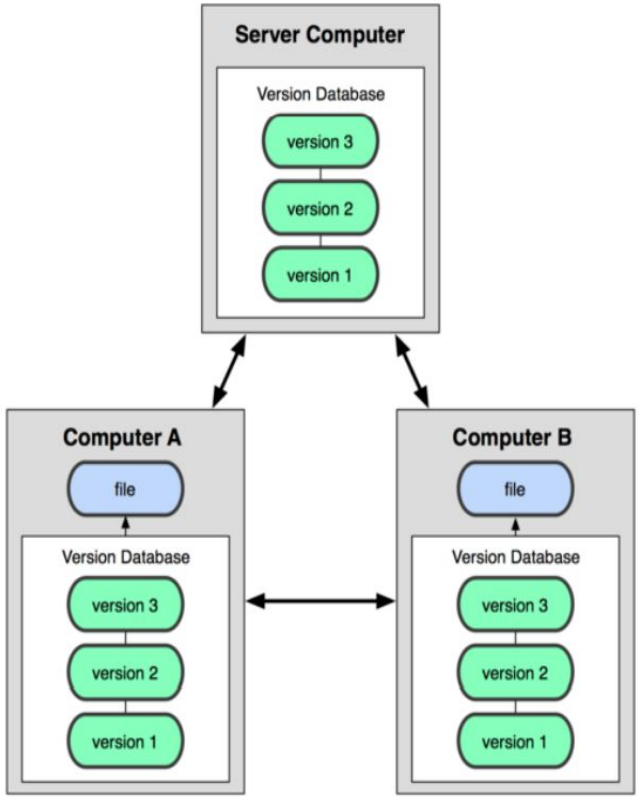


- Highly dependent on the server.
  - Anything goes wrong ther development halts till the server is up again.
  - May even lose entire history or repository if no backups configured.

Distributed VCS

- Entire repository is mirrored locally that includes the full change history.
- Centralised repository hosting possible but complete dependency is not on it.
- Basically everyone owns their own local Copies of the repository.

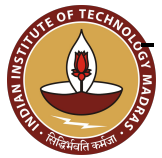
Git is one of the most popular distributed VCS.



Git

- Free and open source version control system.
- Originally authored by Linus Torvalds in 2005 for development of Linux kernel.

- Focus is on speed, data integrity and working on multiple tasks simultaneously.
  - Entire repository is mirrored. Multiple copies, you achieve **Data Integrity**. Very hard to lose data.
  - Nearly every operation is local. You achieve **speed.**
  - Can have multiple branches. **Parallel development.**



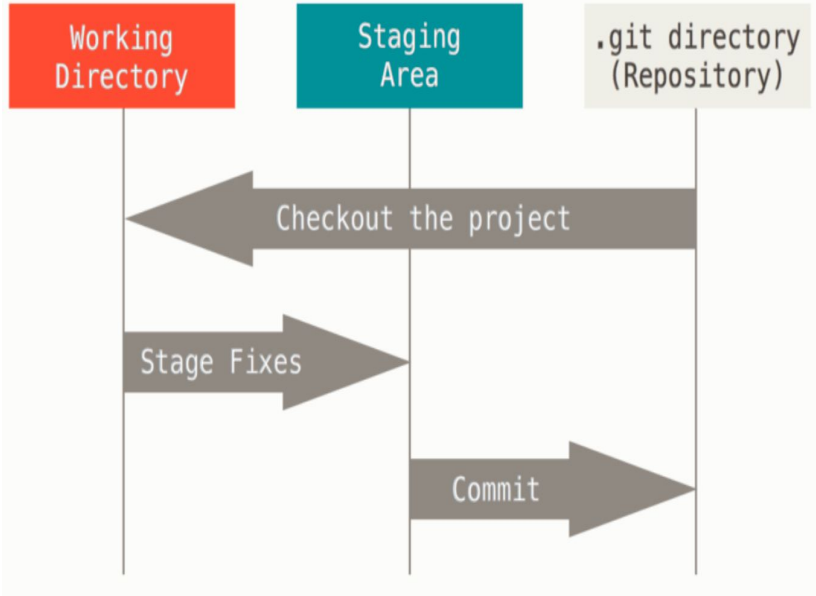
Using Git

Several ways you can use Git

- Command line (Install git on your machine)
- With IDEs.
- GitHub portal using GUI.

Git - Workflow (Local)

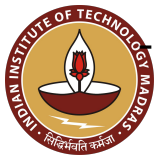
- Three stages the file can be in
  - **Modified, Staged** or **Committed**
- Modified - After you change the file in your working directory.
- Staged - You marked the file to be committed.
- Committed - Changed stored in the local database. But only the marked files(staged) will be committed even if there are more files that were modified.



Git Commands - Getting Started

Either

- Create new repository
  - ***git init*** : Initialize a new git repository in your current directory.
  - Create/change some files in the initialized directory. Now you have new/modified files.
  - ***git add filename***: Stage the modified/new files.
  - ***git commit -m "change description"***: Commit the staged files to DB
- Clone an existing repository.
  - ***git clone url <local_directory>***: Mirror repo located at "url" to "local_directory".
  - Then "*git add*" and "*git commit"* as above.
  - ***git push***: To push changes to the remote repo



Git - Commands

- While you are working on a local copy somebody else working on the same branch updated the remote repo by pushing their changes to it.

?

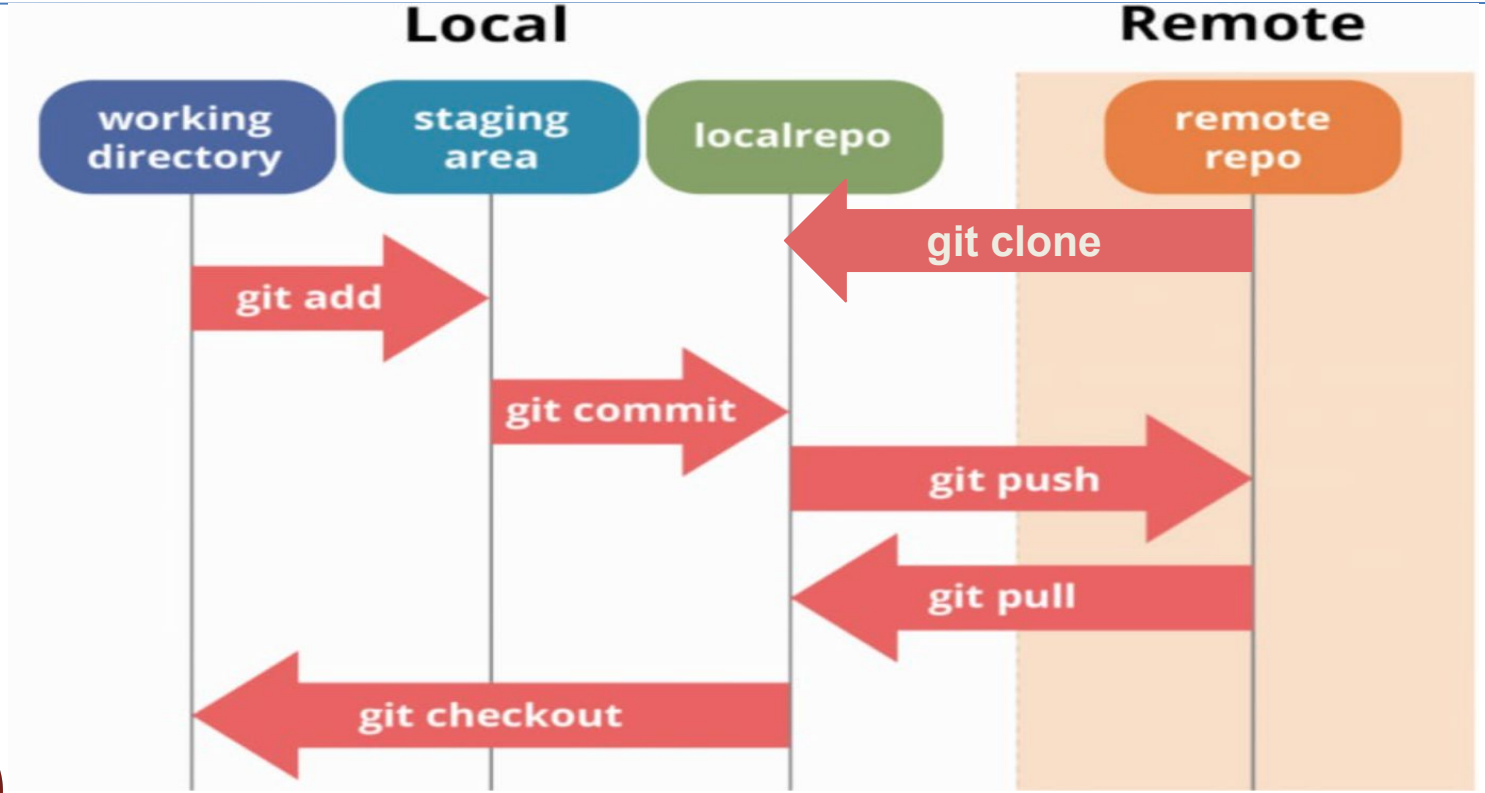
git pull - fetch and download changes from remote repo and update the local rep.

Conflicts? Visit later.

More git commands

- ***git status*** : View the status of your local files in the working directory and staging area.
- ***git diff*** : Show changes between commits, commit and working tree, etc
- ***git reset HEAD <filename>***: unstage the file.
- ***git checkout <filename>***: undo the changes that are committed but not yet pushed to remote
- ***git log*** : Check commit logs/history

Git - Workflow (local + remote)



Git workflow

The workflows described involved only one branch.

Each developer can chose to create multiple branches and work on several tasks simultaneously.

Still their will be some place that they would all want to combine their changes to bring them together.

So we have the concept of **branching** and **merging**.

Git Branching

- Branches are independent versions of repository. Mechanism to diverge work from the main project line.
- Technically it is a pointer to a commit/snapshot.
- Default branch in Git is named as ‘master’. Created at initialization.
- Can create multiple branches to work on several tasks parallelly.
- A developer can switch between branches on the fly..
- ***git branch feature1***: Create a new branch named *feature1* from the base branch you are working on.
- ***git branch*** : List all local branches.
- git checkout : Command to switch branch
  - ***git checkout master:*** switch to master branch.
  - ***git checkout feature1***: switch to *feature1* branch.

Merging

- *A way to combine changes made through one or more branches to a single branch*
- *Basically a way of combining multiple lines of development into a common line.*

- ***git merge feature1***: merge the branch “feature1” to the current working branch.

Merging

- You want to work on a new user story.
- Created a branch named *feature1* from the branch named master.
- Then make the required changes to the files.
- Merge these changes to master.

- You can work parallelly on several tasks if you create multiple branches.

Example

```
$ git branch feature1
// Made changes to file x.py
$ git add x.py
$ git commit -m “change x.py”
$ git checkout master
$ git merge feature1
```

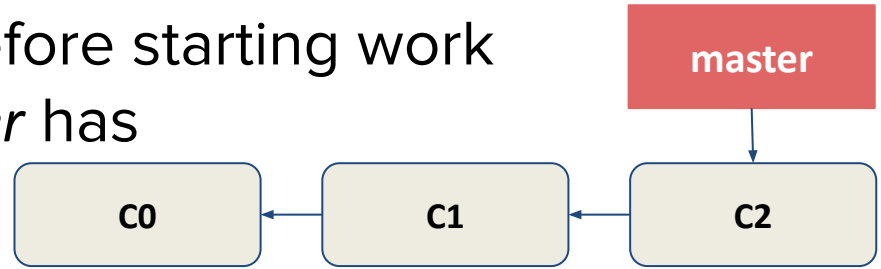
Branching and Merging

Scenario

1. You created a new branch ***feature1*** to work on a new user story.
2. You make some changes to branch ***feature1***.
3. A priority issue was reported and assigned to you to fix immediately.
4. You create another branch named ***issueP*** to fix this issue.
5. You make changes to fix the issue, and merge the branch ***issueP*** to ***master***.
6. You switch back to branch feature1 and resume work on your user story.

Branching and Merging Workflow

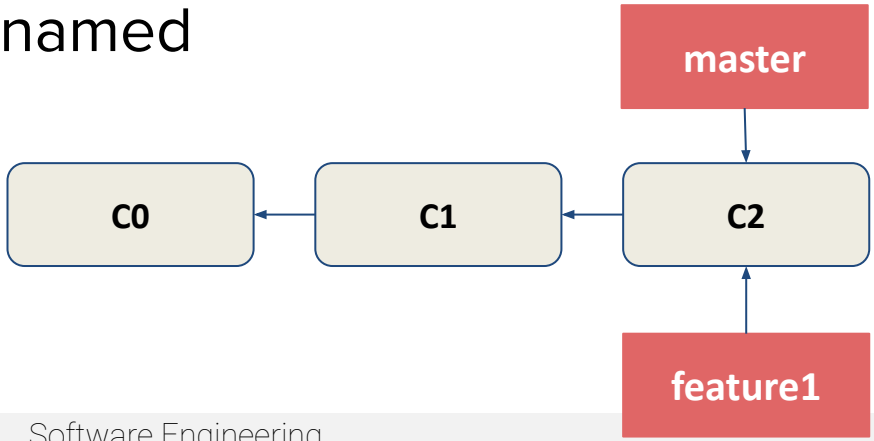
Current branch status before starting work on your user story. *master* has some commits



You created new branch named

feature1

git checkout -b feature1

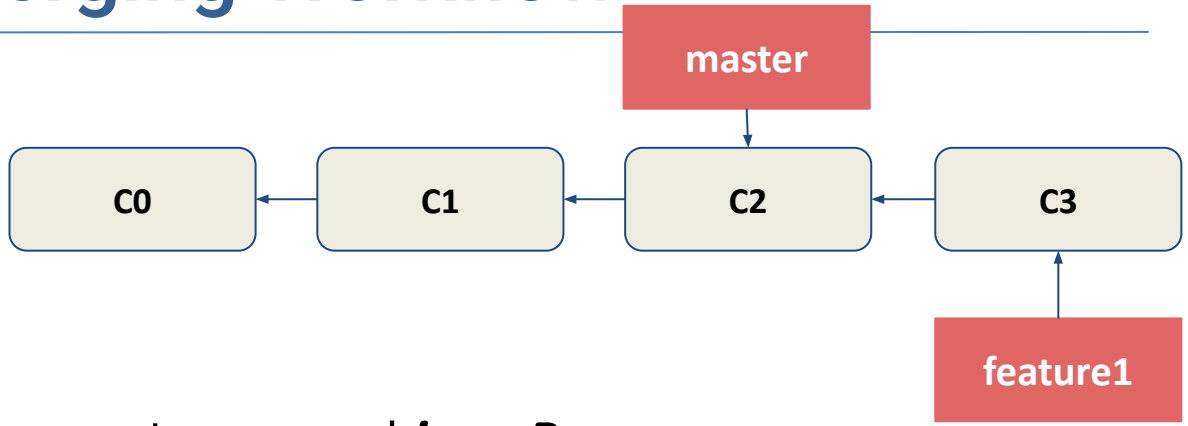


Branching and Merging Workflow

Make change to ***feature1***

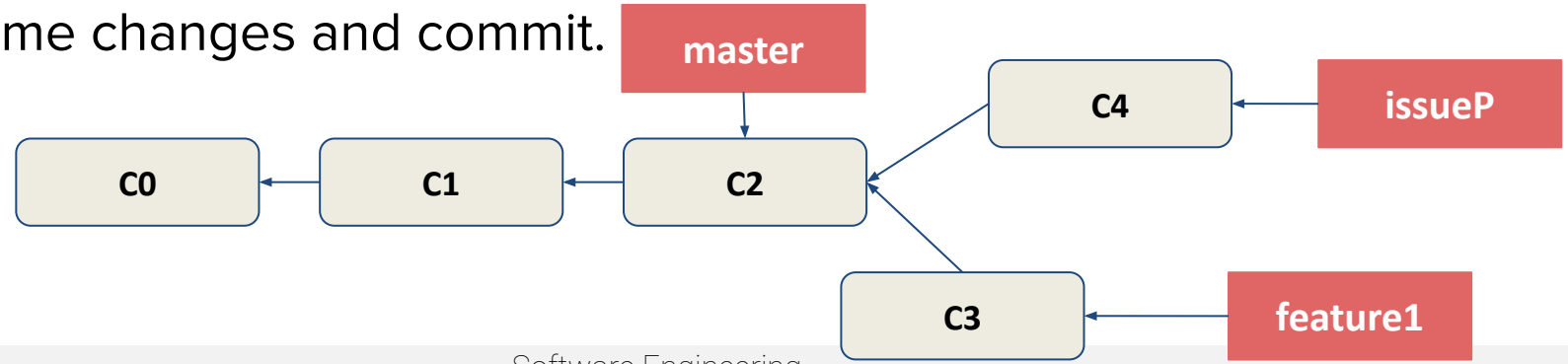
And Commit changes

git commit -m “fix feature1”



Create another branch from master named ***issueP***

Make some changes and commit.



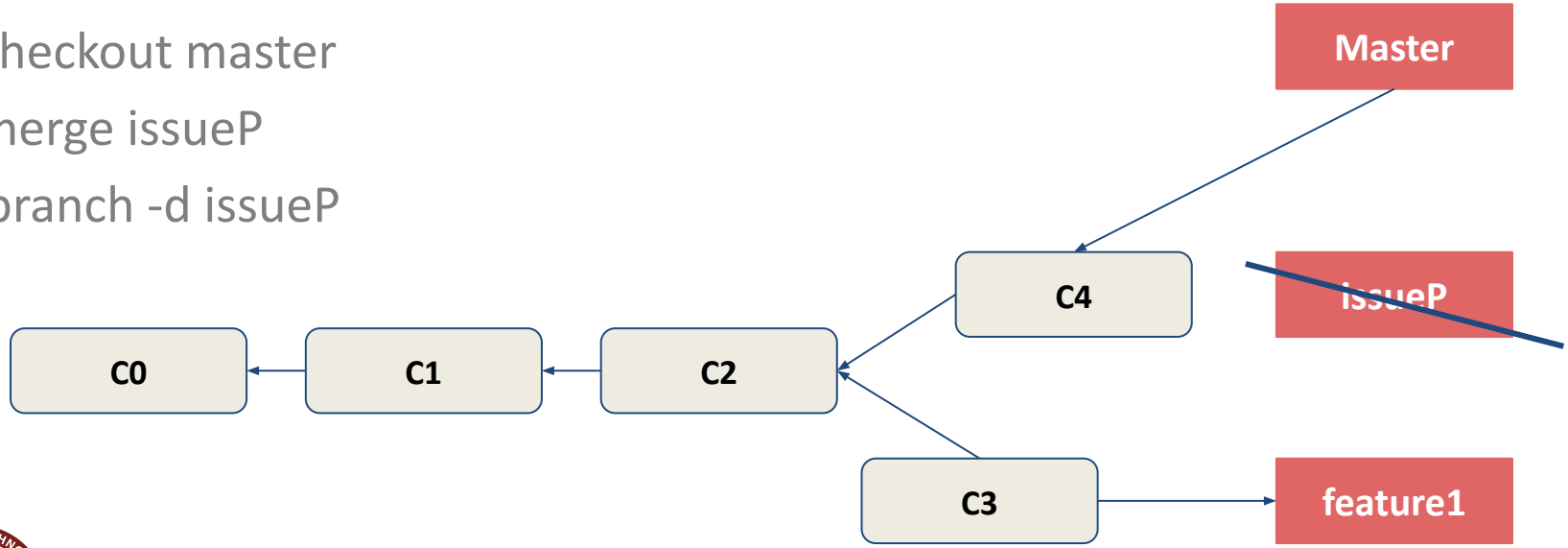
Branching and Merging Workflow

Then merge the priority branch to *master* and delete it

git checkout master

git merge issueP

Git branch -d issueP



Branching and Merging Workflow

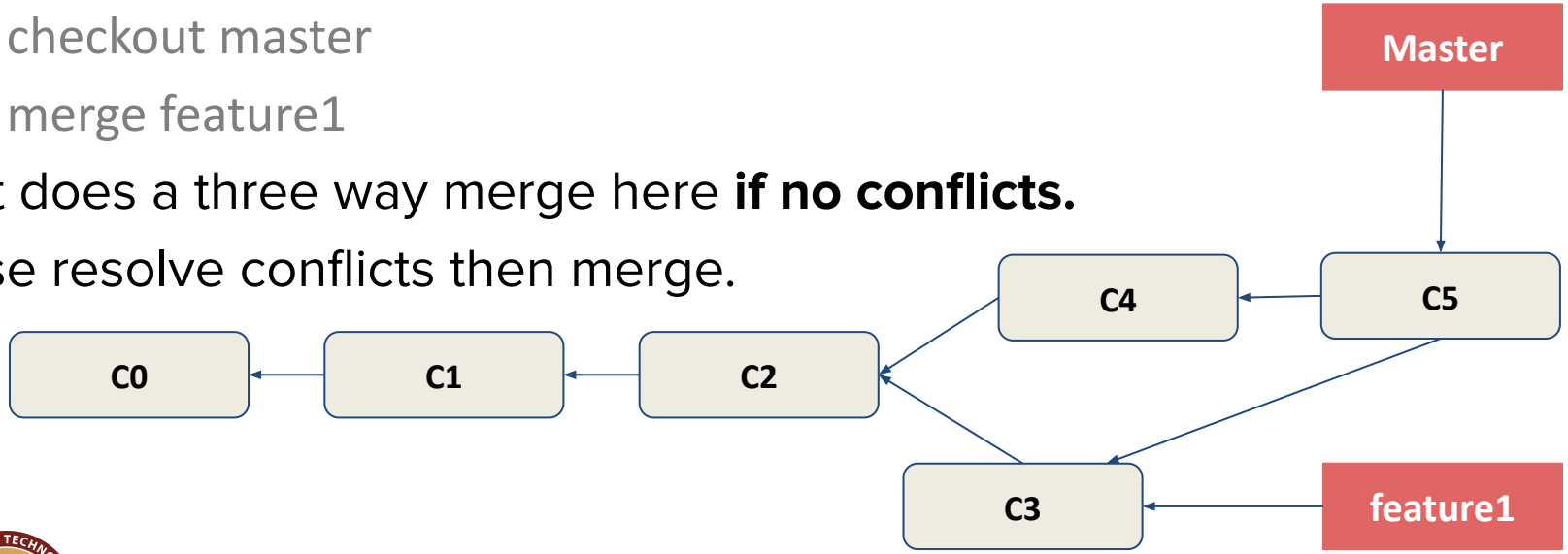
Now master is ahead of branch *feature1* by one commit, lets merge *feature1*

git checkout master

git merge feature1

Git does a three way merge here **if no conflicts.**

Else resolve conflicts then merge.



Git Rebase

- Incorporate changed from one branch to another.

Scenario

1. You created a new branch ***feature1*** to work on a new user story.
2. You make some changes to branch ***feature1***.
3. A priority issue was reported and assigned to you to fix immediately.
4. You create another branch named ***issueP*** to fix this issue.
5. You add a function *fun1* to the code to fix the issue, and you merge the branch ***issueP*** to ***master***.
6. You switch back to branch feature1 and resume work on your user story.
    - But you realize that you were doing something wrong and now you need to call the function *fun1* in your user story implementation i.e in *feature1* branch.

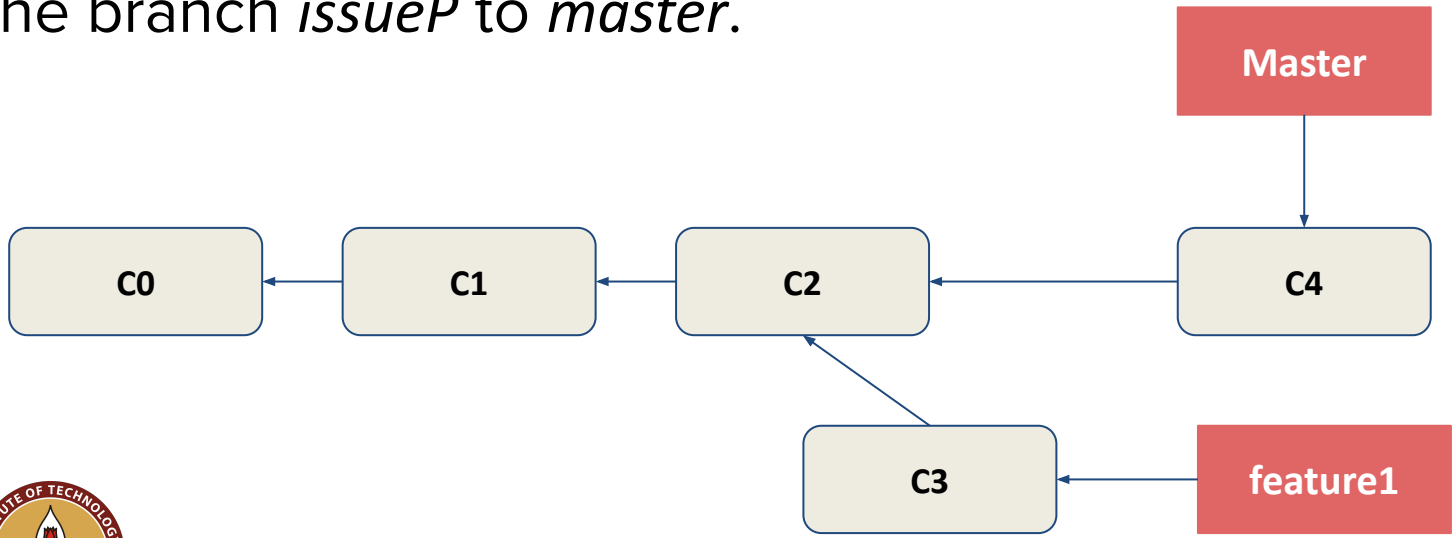
Git Rebase

- But the changes made through branch *issueP* are not available in the branch feature1 as it is based on a commit that was before merging the branch *issueP* to master.
- And to test your changes locally you need to have the function ***fun1()*** in your code base of branch *feature1*.
- Solution **git rebase.**
- Rebase branch *feature1* to latest master branch commit.
- Rebasing can be used to combine new commits to your working branch. Basically changing the base commit of the branch *feature1*.
- There could be conflicts while rebasing if same files are changed in your changes and the new commits being integrated. Resolve them to complete rebase.

Git Rebase Workflow

Workflow till step 5 is same as in the the previous scenario.

So let's Consider this state(after step 5), which is after merging the branch *issueP* to *master*.

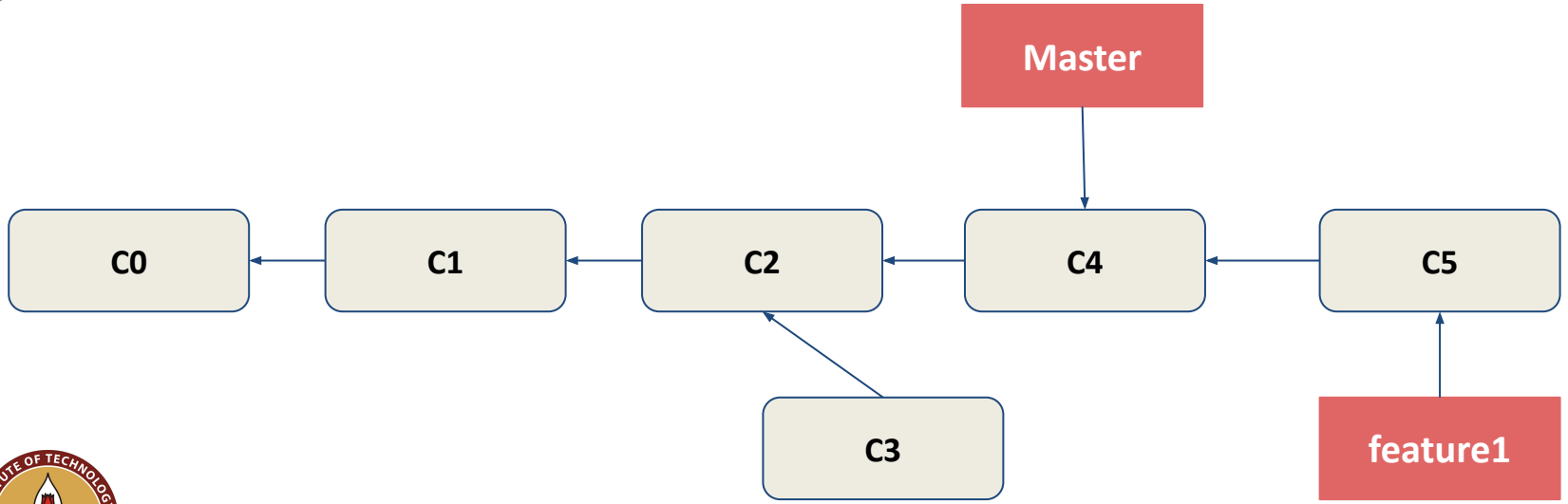


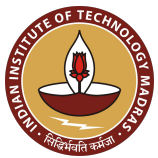
Git Rebase Workflow

Rebase your branch to the latest commit.

git checkout feature1

git rebase master



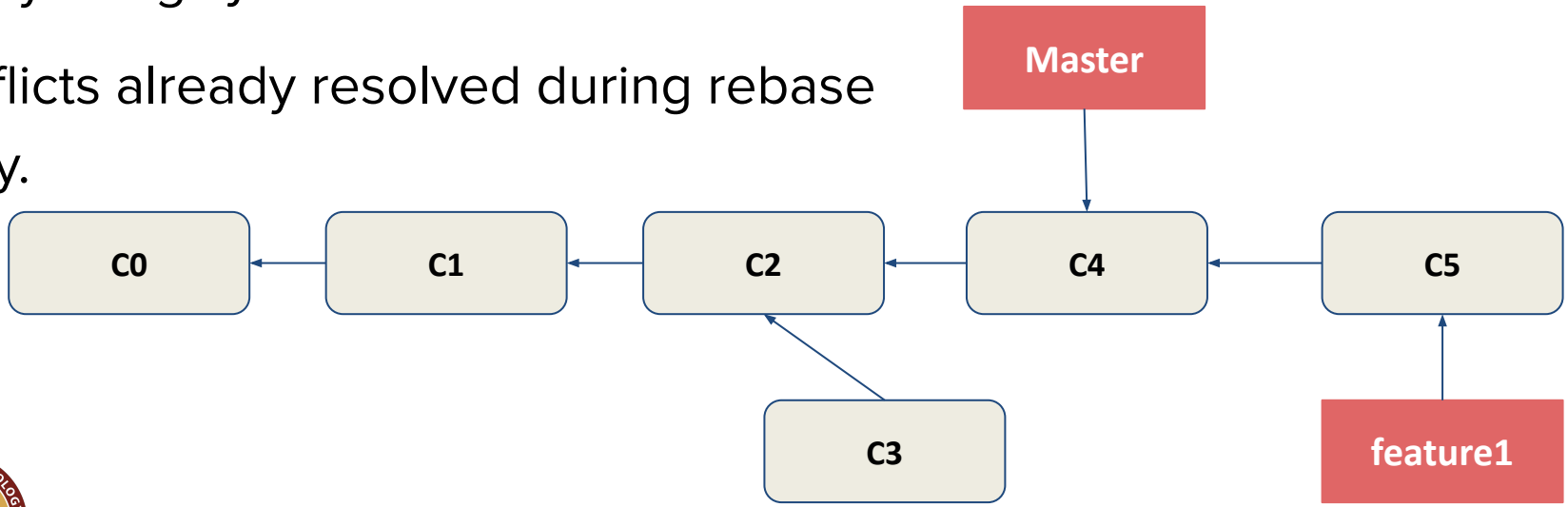


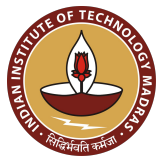
Git Rebase Workflow

Now call *fun1()* in branch *feature1*, make Changes, test your changes.

Finally merge *feature1* to *master*.

Conflicts already resolved during rebase If any.





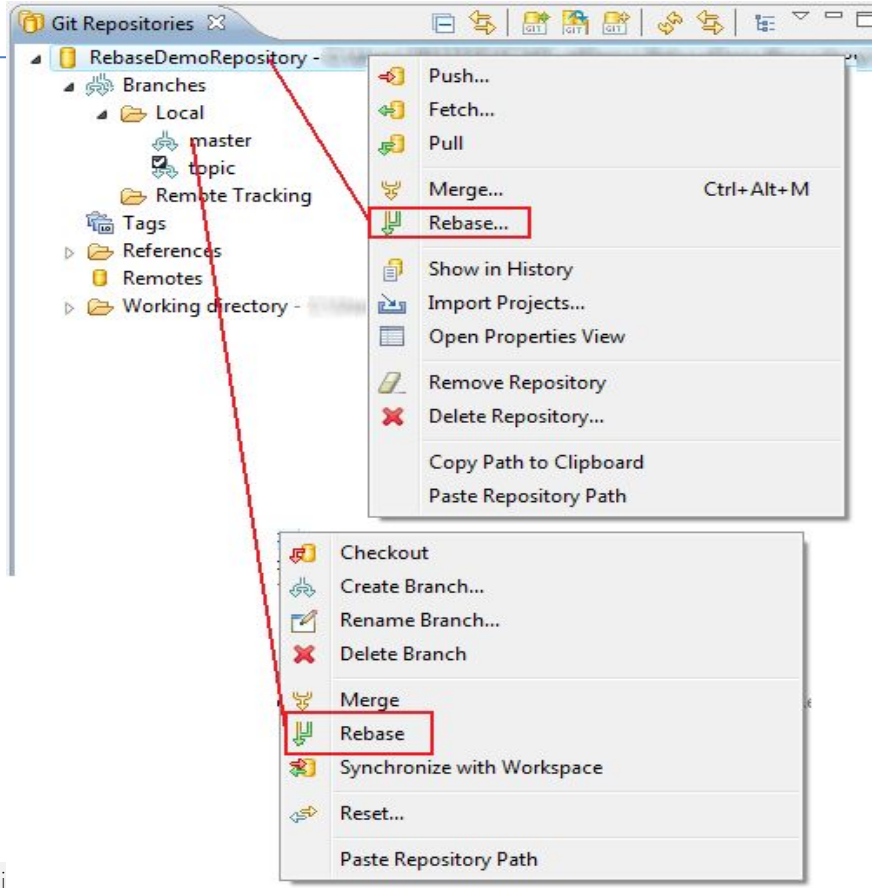
Workflow Reference

- For more details on git refer

  https://git-scm.com/book/en/v2

Git on IDEs

- Almost all popular IDEs support Git.
- Either integrated by default or through plugins.
- Intuitive if Git workflow is known.



GitHub

- Online git repository hosting service.
- Provides web-based graphical user interface to manage your git project created on GitHub.

- Also provides
  - Access control, bug tracking, software feature requests, task management, continuous integration and wikis

GitHub tutorial

https://docs.github.com/en/get-started/quickstart/hello-world