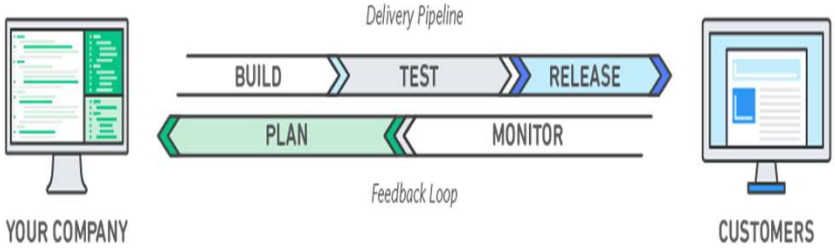
DevOps

Ameya J. Joshi
Email : ameya.joshi_official@outlook.com
Cell no. : +91 9850676160

1



2

DevOps Benefits

* Speed
* Rapid Delivery
* Reliability
* Scale
* Improved Collaboration
* Security

3

DevOps Best Practices

* Continuous Integration
* Continuous Delivery
* Microservices
* Infrastructure as Code
* Monitoring and Logging
* Communication and Collaboration

4

DevOps Tools

* GIT – DVCS
* Jenkins – CI/CD
* Docker – Containerization
* SpringBoot – Implement Microservices
* AWS – Cloud Deployment Platform

5

GIT

6

What is GIT

* GIT was founded by Linus Torvalds. It is open source software.
* It is a Distributed Version Control System
  * This simply means, instead of one remote central repository to manage and control your code, you will also have local repositories on your local installations which will synchronize with remote repository.
* GIT implementations :
  * GitHub
  * Bitbucket

7

A Brief History of Git

* Linus uses BitKeeper to manage Linux code
* Ran into BitKeeper licensing issue
  * Liked functionality
  * Looked at CVS as how not to do things
* April 5, 2005 - Linus sends out email showing first version
* June 15, 2005 - Git used for Linux version control

8

Git is Not an SCM

Never mind merging. It's not an SCM, it's a distribution and archival mechanism. I bet you could make a reasonable SCM on top of it, though. Another way of looking at it is to say that it's really a content-addressable filesystem, used to track directory trees.

Linus Torvalds, 7 Apr 2005

http://lkml.org/lkml/2005/4/8/9

9

Centralized Version Control

* Traditional version control system
  * Server with database
  * Clients have a working version
* Examples
  * CVS
  * Subversion
  * Visual Source Safe
* Challenges
  * Multi-developer conflicts
  * Client/server communication

10

Distributed Version Control

* Authoritative server by convention only
* Every working checkout is a repository
* Get version control even when detached
* Backups are trivial
* Other distributed systems include
  * Mercurial
  * BitKeeper
  * Darcs
  * Bazaar

11

Git Advantages

* Resilience
  * No one repository has more data than any other
* Speed
  * Very fast operations compared to other VCS (I'm looking at you CVS and Subversion)
* Space
  * Compression can be done across repository not just per file
  * Minimizes local size as well as push/pull data transfers
* Simplicity
  * Object model is very simple
* Large userbase with robust tools

12

Some GIT Disadvantages

* Definite learning curve, especially for those used to centralized systems
  * Can sometimes seem overwhelming to learn
* Documentation mostly through man pages
* Windows support can be an issue
  * Can use through Cygwin
  * Also have the msysgit project

13

Installing GIT

* GIT
  * URL : https://git-scm.com/downloads
* Cmder
  * URL : http://cmder.net/
  * The full version of the command line emulator comes with pre-insatlled GIT
* Atom :
  * URL : https://atom.io/
  * A text editor we will use during training

14

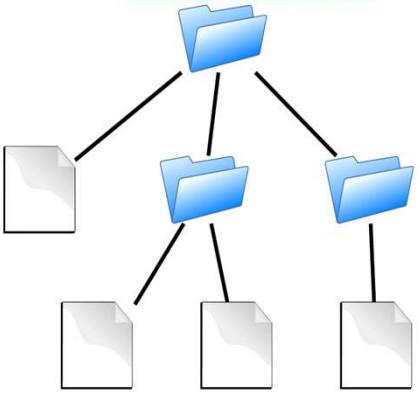
Installing GIT

* Ubuntu Linux :
  * sudo apt update
  * sudo apt-get install git
  * Provide the root password
  * whereis git

15

GIT Architecture

* Source code contains
  * Directories
  * Files
* It is the substance of a software configuration



16

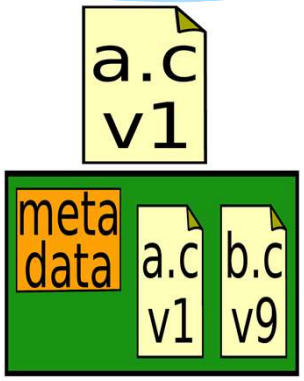
Git Architecture

* Repository

Contains

* files
* commits

* records history of changes to configuration



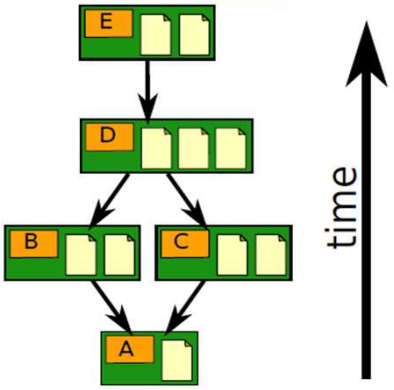
17

Git Architecture

* Repository

Contains

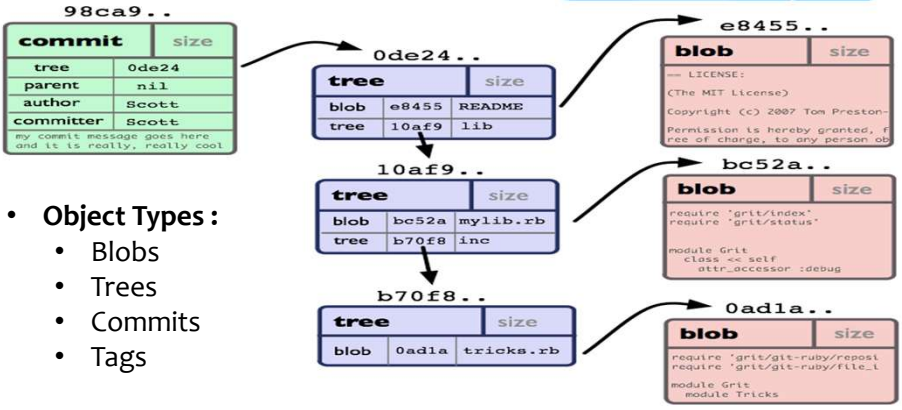
* files
* commits
* ancestry relationships
* form a directed acyclic graph (DAG)



18

Git Architecture

Git Object Model



- **Object Types :**
  - Blobs
  - Trees
  - Commits
  - Tags

19

Git Architecture

- Index
  - Stores information about current working directory and changes made to it
- Object Database
  - Blobs (files)
    - Stored in .git/objects
    - Indexed by unique hash
    - All files are stored as blobs
  - Trees (directories)
  - Commits
    - One object for every commit
    - Contains hash of parent, name of author, time of commit, and hash of the current tree
  - Tags

20

How GIT Works

* Repositories (Repos)
  * A repo is a container for a project you want to track with GIT
  * You may have many different repos for many different projects on your computer.
  * It is like a "project folder that GIT tracks the contents of for us"

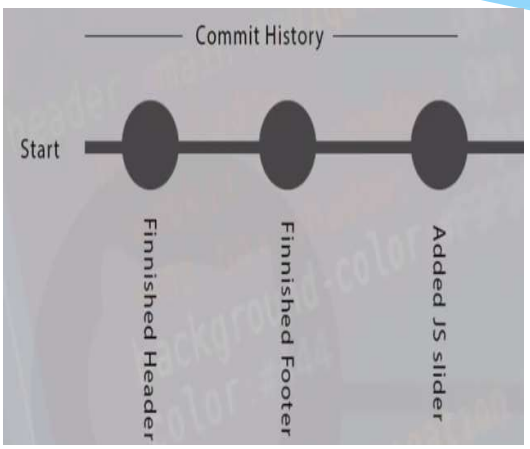
21

Repos

* My Project folder here acts as an repo
* This is because of .git folder within My Project folder.
* Because this .git file is in My Project git is going to track changes to all files, folders, sub-folders from within My Project folder.

22

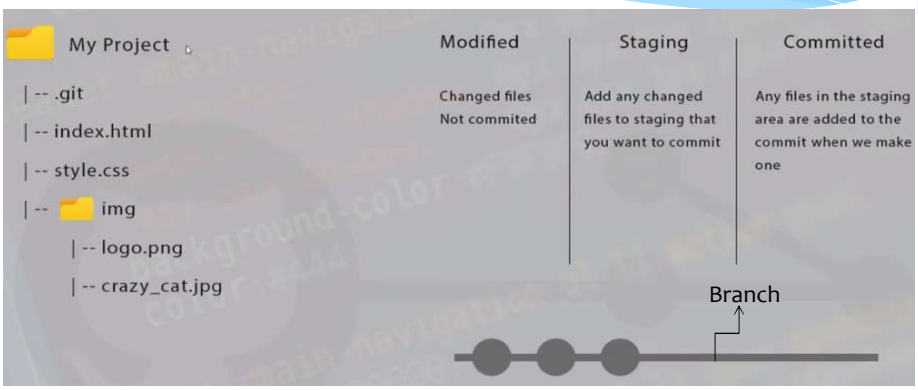
Commits



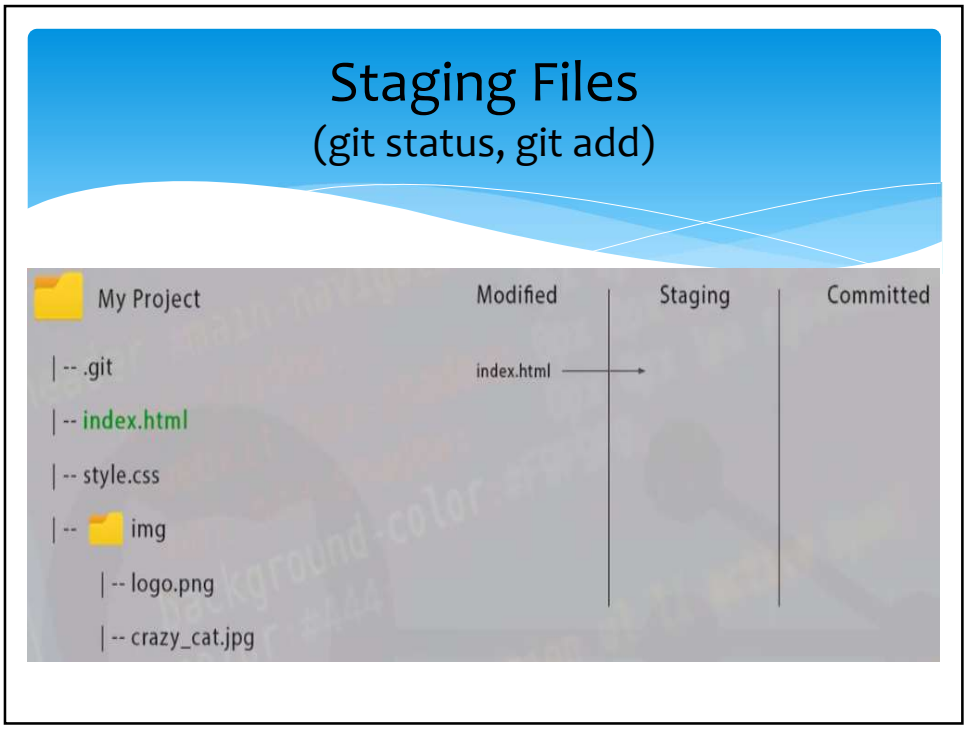
- In the simplistic form commits are like savepoints.
- You can commit the changes you have done to your projects periodically.
- Each commit might describe a milestone in your project or some phase in your project.
- The committed code can also be rolled back if so desired

23

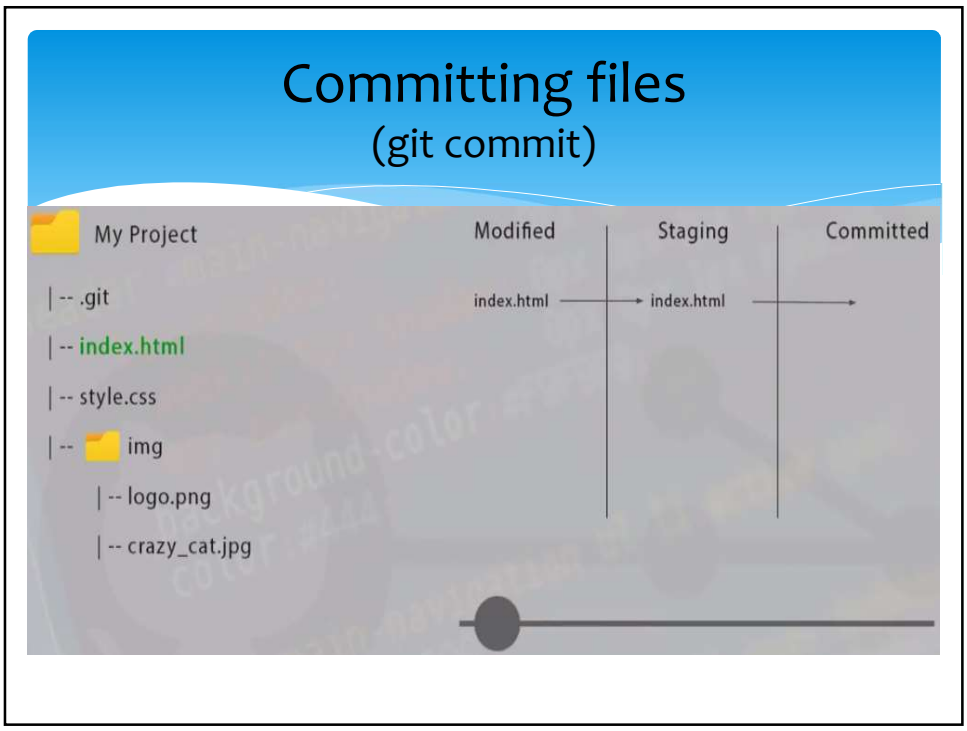
Stages and Commits



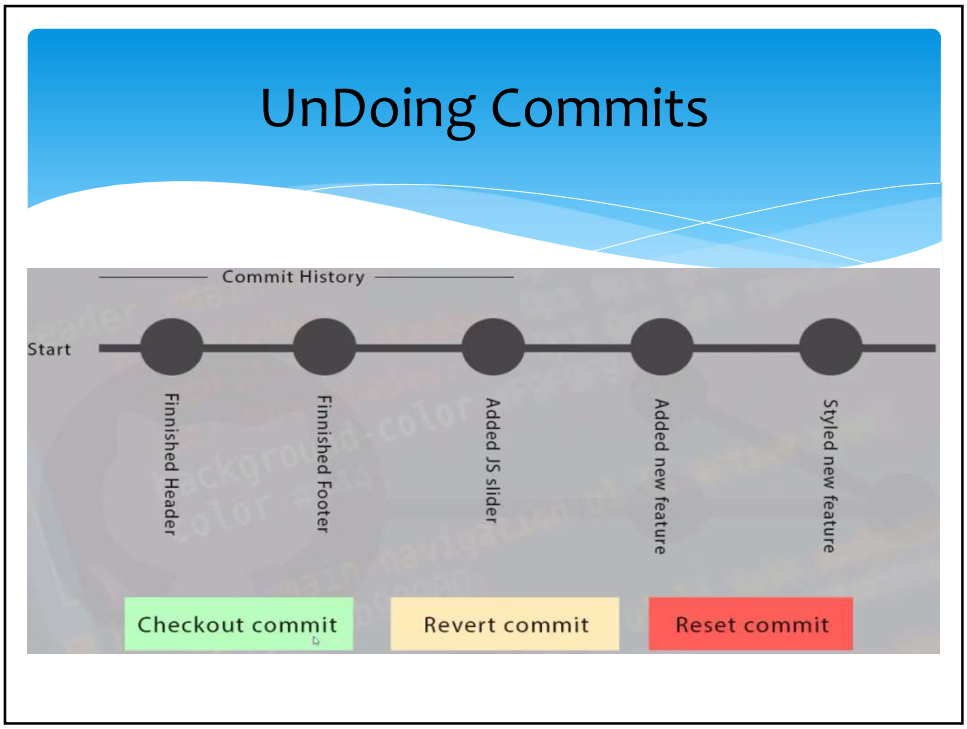
24


Staging Files
(git status, git add)
My Project
| -- .git
| -- index.html
| -- style.css
| -- img
| -- logo.png
| -- crazy_cat.jpg
Modified
Staging
Committed
index.html

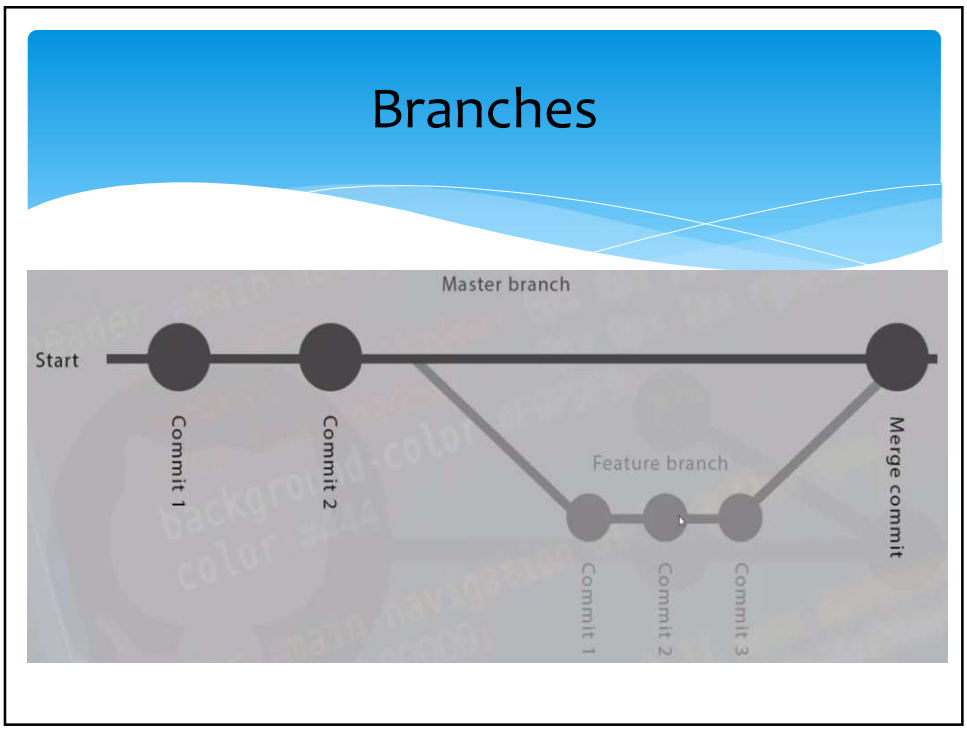
25


Committing files
(git commit)
My Project
| -- .git
| -- index.html
| -- style.css
| -- img
| -- logo.png
| -- crazy_cat.jpg
Modified
Staging
Committed
index.html
index.html

26

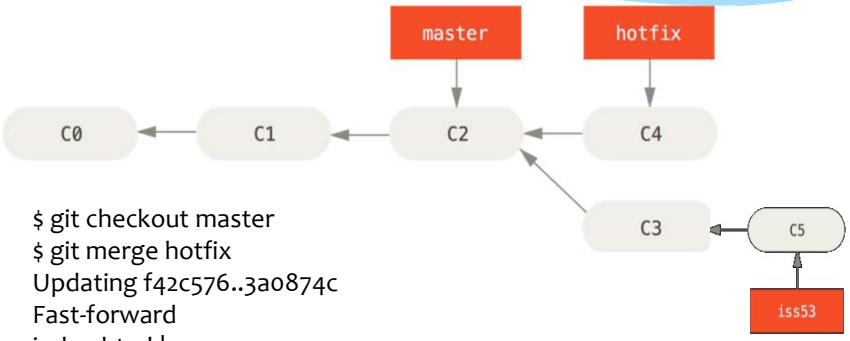

UnDoing Commits
Commit History
Start
Finnished Header
Finnished Footer
Added JS slider
Added new feature
Styled new feature
Checkout commit
Revert commit
Reset commit

27


Branches
Master branch
Start
Commit 1
Commit 2
Feature branch
Commit 1
Commit 2
Commit 3
Merge commit

28

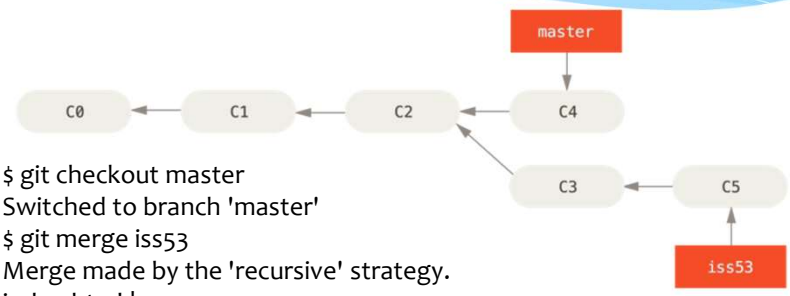
merge




```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
index.html | 2 ++
1 file changed, 2 insertions(+)
```

29

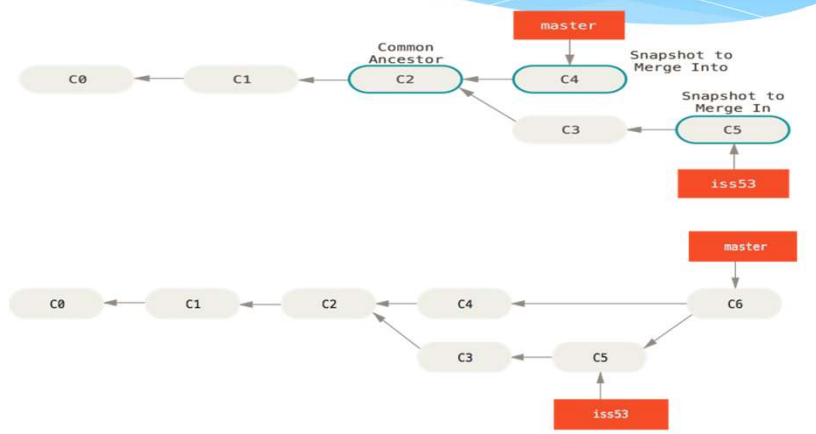
merge




```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
Merge made by the 'recursive' strategy.
index.html | 1 +
1 file changed, 1 insertion(+)
```

30

merge



31

GitHub (BitBucket)

* Github is a service that lets us set up hosted repositories
* Central online repository which multiple team members could access

32

GitHub (BitBucket)

* URL : https://github.com/
* Signup
* Verify email
* Login

33

GitHub (BitBucket)

* Pushing the projects to remote repositories
  * Push to remote from existing local repo
    * git push
  * Push to remote where you do not have any existing local repo
    * git clone
    * git push

34

GitHub (BitBucket)

* Synchronizing the local and remote repositories
  * git pull
* In the remote repository
  * Create the pull request
  * Merge/confirm the pull request

35

GitHub (BitBucket)

* Fork and Contributing
* If you want to modify and contribute code to some different account external repository, it would seem simple just to pull the repository to your local environment modify it and then push to remote repository.
* However this won't be possible most times as you will probably not have any access to pull from this repository.
* The solution to this is using fork.

36

GitHub (BitBucket)

* Fork sort of creates a copy of the repository from the other account to your account
* Then you can clone the repository to your local environment.
* Make the contribution by adding/modifying your code
* Stage commit your changes and push it to remote repository
* Create the pull request.
* The owner of the other account can then merge the modified repository.

37

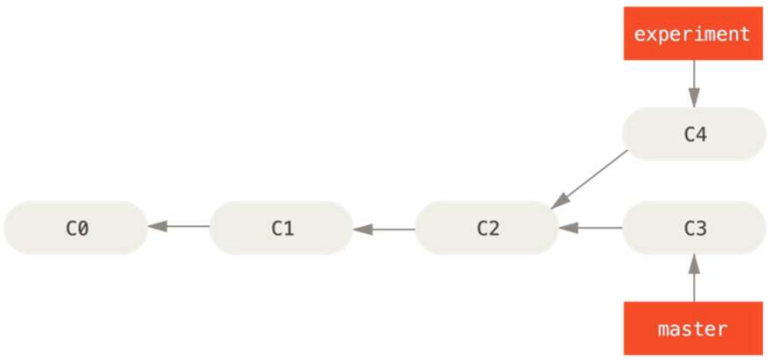
rebase

* In Git, there are two main ways to integrate changes from one branch into another: the merge and the rebase.

38

rebase

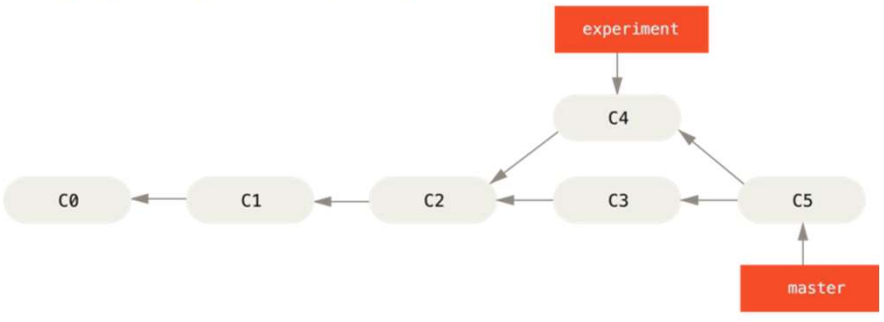
* Scenario



39

rebase

* The easiest way to integrate the branches, as we've already covered, is the merge command. It performs a three-way merge between the two latest branch snapshots (C3 and C4) and the most recent common ancestor of the two (C2), creating a new snapshot (and commit).



40

rebase

* However, there is another way: you can take the patch of the change that was introduced in C4 and reapply it on top of C3. In Git, this is called *rebasing*. With the rebase command, you can take all the changes that were committed on one branch and replay them on a different branch.

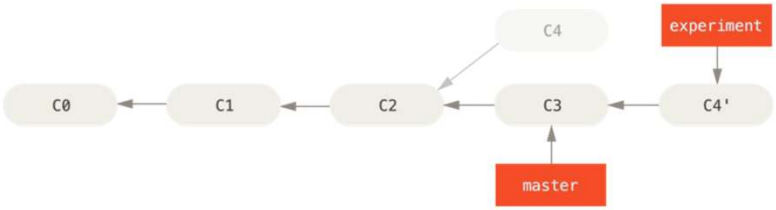
* Eg.

$ git checkout experiment
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: added staged command

41

rebase

* This operation works by going to the common ancestor of the two branches (the one you're on and the one you're rebasing onto), getting the diff introduced by each commit of the branch you're on, saving those diffs to temporary files, resetting the current branch to the same commit as the branch you are rebasing onto, and finally applying each change in turn.

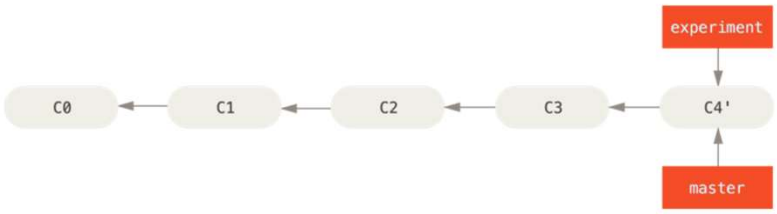


42

rebase

$ git checkout master

$ git merge experiment



43

cherry-pick

* Cherry-picking
* Apply the changes introduced by some existing commits
* It might happen that In your source code on your branch you are facing some problem, the same problem has however been fixed by a teammate on some other branch and code has been committed on that branch.
* Cherry-pick enables you to pick this commit and perform a commit on your branch.
* So essentially you apply changes introduce by some other existing commits.

44

Cleaning Up

* git fsck
  * Checks object database to make sure all is sane
  * Can show information about dangling objects
* git gc
  * Cleans up repository and compress files
  * When used with --prune, cleans out dangling blobs
  * Can really speed up larger repositories

45

git hooks

* Git hooks are scripts which trigger when you perform specific actions in GIT.
* They are useful to automate the tasks
* Example You can create a git hook to run some other program or do some other task everytime you commit your code.

46

git hooks

Client side and Server side Hooks

* GIT groups hooks into two categories.
* Client side hooks are triggered by operations like commit and merger
* Server side hooks are triggered by network operations like receiving pushed commits etc.

47

git hooks

* **The git hooks folder**
* Upon initializing a repo, you will have a folder hooks within your .git folder.
* Within this you will have some sample hooks
* These scripts are all shell and perl scripts
* However The Docs claim that you could use any scripting language as long as the script is “executable”.

48

.gitignore file

* Git sees every file in your working copy as one of three things:
* tracked - a file which has been previously staged or committed;
* untracked - a file which *has not* been staged or committed; or
* ignored - a file which Git has been explicitly told to ignore.

49

.gitignore file

* Ignored files are usually build artifacts and machine generated files that can be derived from your repository source or should otherwise not be committed. Some common examples are:

* dependency caches, such as the contents of /node_modules or /packages
* compiled code, such as .o, .pyc, and .class files
* build output directories, such as /bin, /out, or /target
* files generated at runtime, such as .log, .lock, or .tmp
* hidden system files, such as .DS_Store or Thumbs.db
* personal IDE config files, such as .idea/workspace.xml

50

.gitignore file

- Ignored files are tracked in a special file named .gitignore that is checked in at the root of your repository.
- There is no explicit git ignore command: instead the .gitignore file must be edited and committed by hand when you have new files that you wish to ignore.
- .gitignore files contain patterns that are matched against file names in your repository to determine whether or not they should be ignored.

51

.gitignore file

```
# Compiled source #
###################
*.com
*.class
*.dll
*.exe
*.o
*.so
# Packages #
############
*.7z
*.dmg
*.gz
*.iso
*.jar
```

52