6.2 Git Architecture

6.2 Git Architecture

Installing Git
One of the foremost popular VCS tools in use today is Git. Git is a Distributed VCS, a category referred to as DVCS. Like many of the foremost popular VCS systems available today, Git is free and open source.

Mac OS - Command-line tools will install Git by default: https://Git-scm.com/download/mac
Linux - Use your favourite package manager: https://Git-scm.com/download/linux
Windows - https://Gitforwindows.org/

Download Git

Understanding Git Architecture

A Version Control System usually has three core functionalities. It must be able to store content, track changes to said content (all history including merge metadata), and optionally distribute the content and commit history with project collaborators.

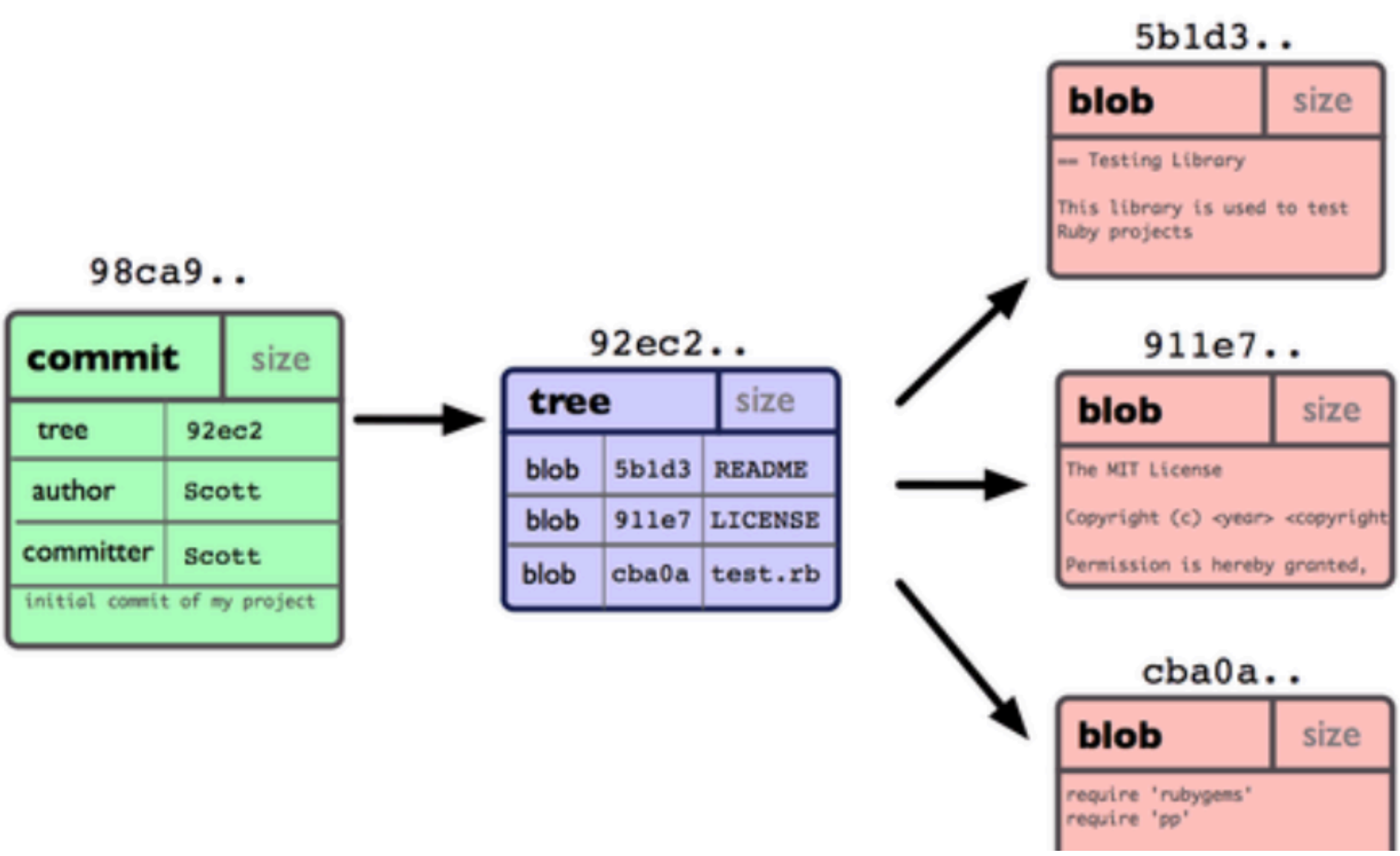
Everything is Local
This means, you do not need to be connected with the server all the time, because 95% of Git commands can be executed in the local repository. Comparing different versions of files, Getting the history of files, Committing the changes to the files - all are LOCAL and FAST in Git.
When you clone a Git repo, as the name suggests, it downloads everything. After that, it does not need anything from the server/peer to continue working. That means you do not need a server at all to work with Git.

Snapshots not diffs
Git keeps snapshots of the file system, not the diffs. This is a major difference from the other systems. This makes it easy to compare different versions and also support features like branching.

Additions only
Almost every operation on Git adds data to the Git database. Since Git rarely deletes any changes, it’s highly unlikely for you to lose any changes.

There are 3 main types of objects in a Git database. They are Blob, Tree and Commit.

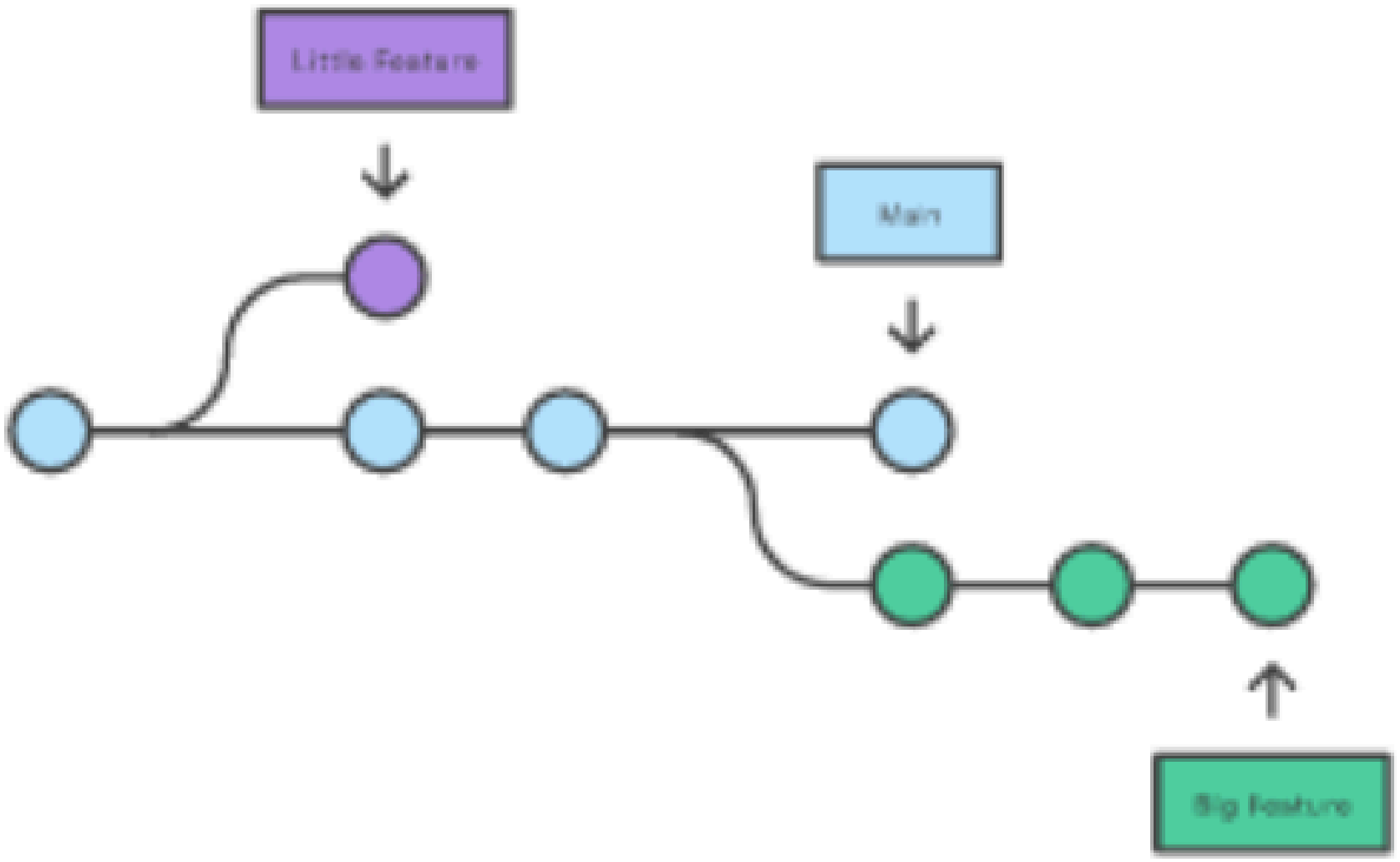
- **Blob**: This object as we have seen above stores the original content.
- **Tree**: This object is used to store directories present in our project.
- **Commit**: This object is created whenever a commit is made and abstracts all the information for that particular commit.



```
module Test
  module Tester
```

Tags
It is a user-friendly name for a hash, Pointer to a Commit. Git can tag specific points in a repository's history as being important. Typically, people use this to mark release points (v1.0, v2.0 and so on).

Branches
In Git, branches are a component of your everyday development process. Branches are the running tags that point to the most recent of a code path. Git branches are a pointer to a snapshot of your changes. When a brand new feature is needed to be added or there is a bug that needs to be fixed, a replacement branch is created to enclose changes. This makes it harder for incompatible code to get merged into the main codebase. It gives the freedom to clean up history before merging it into the most recent branch.



Head
The Head is a special pointer. It points out the last commit in the current checkout branch. It is like a pointer to any reference. The Head can be considered as the current branch as well. When you move from one branch to another with checkout, the Head is transferred to the new branch.

There are 4 possible states for a file in Git.

Untracked
This is a new file and is not tracked in Git yet. You can change the state of these files to staged by using git's add command

Unmodified
These are any files that haven't been modified since the last commit. they're going to still be included in the next commit but remain as is.

Modified
This file/changeset is modified from the last version. These are files that are modified since the last commit. These files are included in the next commit but are going to be included in their respective new form.

Staged
These are files that are either not in the last commit or are modified files that git will include in the next commit using git's add command.

There are three core areas to git. These are the Working Area, the Staging Area (also referred to as Index), and the Repository. When working in a Git repository, files and modifications will transfer from the Working Area to the Staging Area and end up at the Repository.

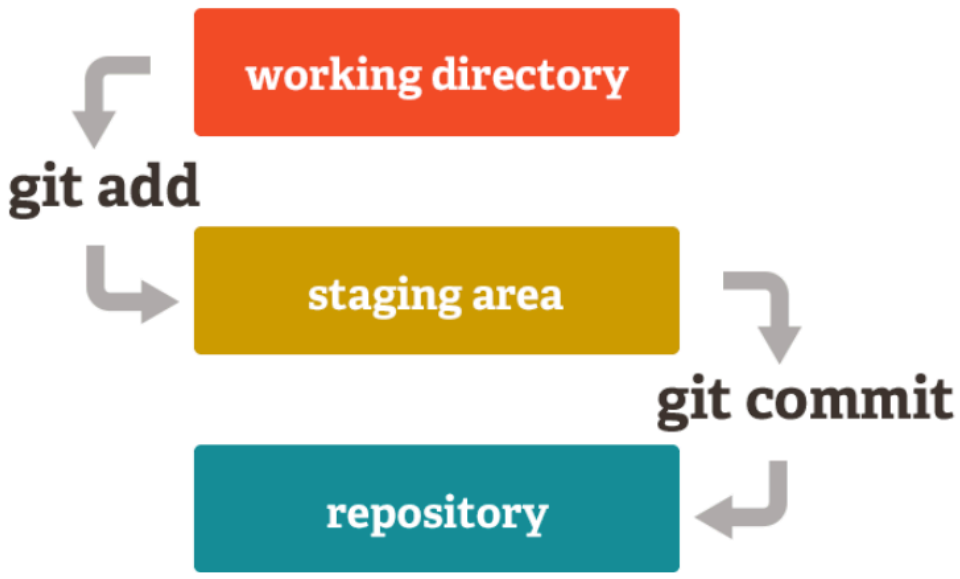
1. Working Area

1. Working Area

The Working area is the area where you're currently working. This area is also referred to as the untracked area of Git. Any changes to files are going to be marked and seen in the Working Tree. Here if you make changes and don't explicitly save them to Git, you'll lose the changes made to your files.

2. Staging Area

The staging area is when git starts tracking and saving changes that occur in files. These saved changes reflect within the .git directory. that's about it when it involves the staging area. These changes here will be committed when you commit next.
If something needs to be committed, first add it to the staging area.



Source: https://miro.medium.com/max/1372/1*diRLm1S5hkVoh5qeArND0Q.png

3. Repository

A repository is a collection of branches.
Branches in remote repos can be addressed as {remote-repo-name}/{branch-name}.
When you checkout to a remote branch, a local branch will be created as well.

Most of the time you would be working with one remote repo (usually called origin) but git architecture does not prevent us from adding multiple remote repos. You can add all your peer repos and collaborate with them without ever touching a single server in a completely distributed manner.

Local Repo

The Local Repository is everything in your .git directory. Mainly what you will see in your Local Repository are all of your checkpoints or commits. It is the area that saves everything.

Remote / Upstream Repo

Remote repositories are versions of your project that are hosted on the Internet or network somewhere. You can have several of them, each of which generally is either read-only or read/write for you.

GET IN TOUCH

University of Moratuwa
Centre for Open & Distance Learning
CODL

011 308 2787/8

011 265 0301 ext. 3850,3851

open@uom.lk

University Website

CODL Website

Data retention summary