



git

By Vijeta Angeer

Agenda

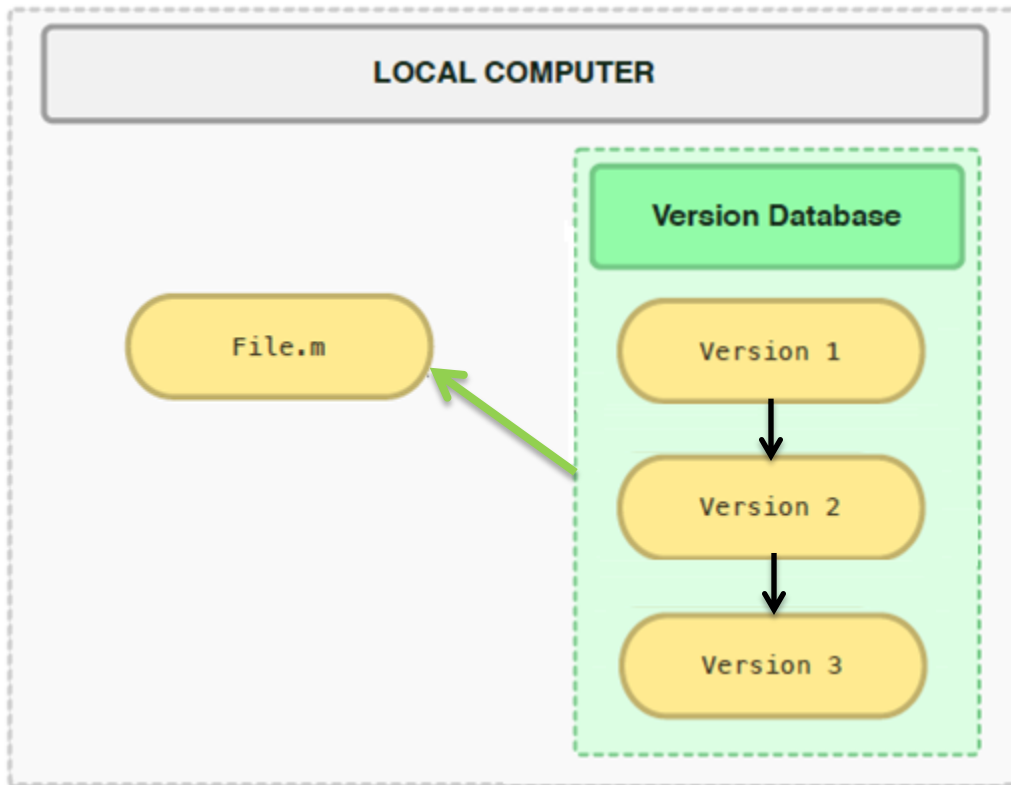
- Version Control Systems
- Git History
- Why Git?
- Getting started
- GIT Architecture
- Branching and Merging
- GIT Workflows



Version Control System

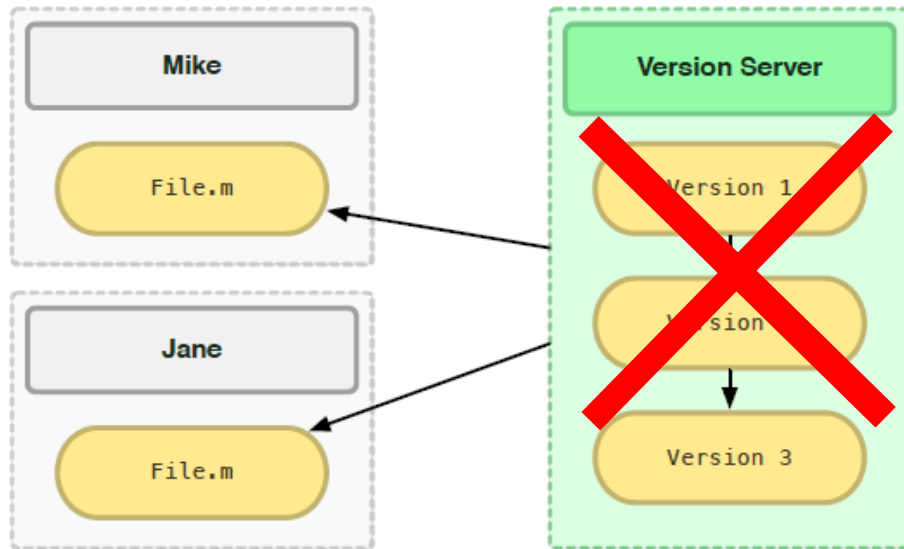
- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
- If you screw things up or lose files, you can easily recover.
- Almost all “real” projects use some kind of version control
- Any company with a clue uses some kind of version control
- Companies without a clue are bad places to work!!!

Centralized Version Control Systems



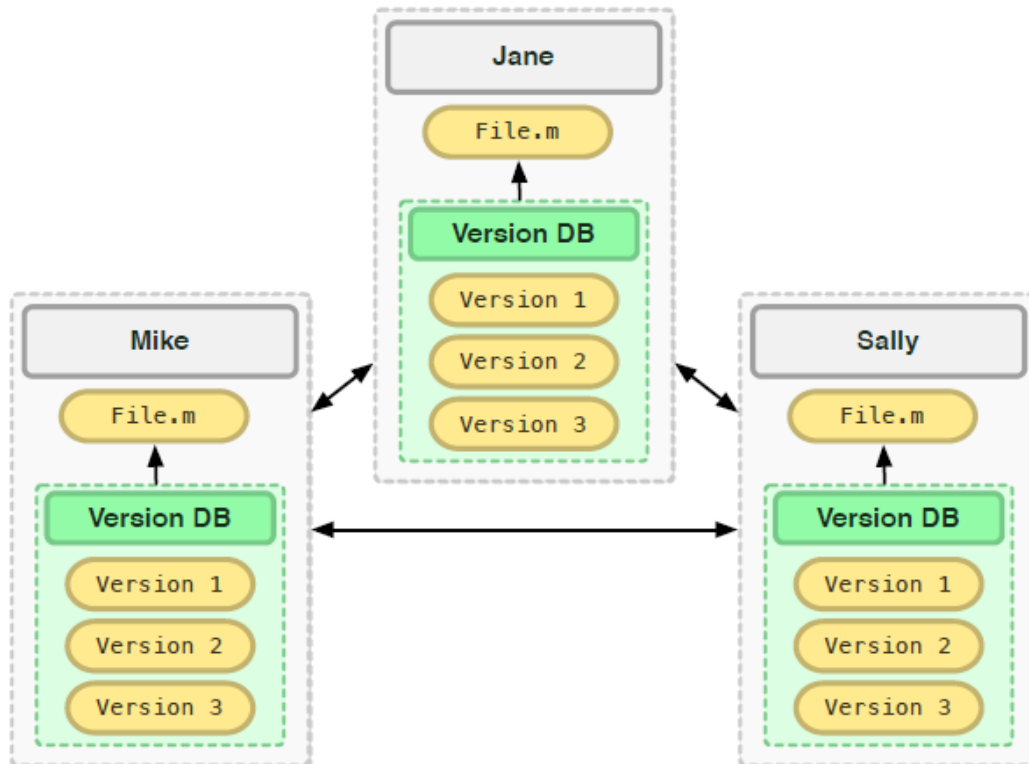
- These systems think of the information they keep as a set of files and the changes made to each file over time.
- Single server that contains all the version files, and a number of clients that check out files from that central place.
- Example
 - CVS
 - Subversion
 - Visual Source Safe

Problems faced in CVCS



- Single point of failure
- Dependent on access to the server
- Hard to manage a server and backups
- It can be slower because every command connects to the server.
- Branching and merging tools are difficult to use.
- Multi-developer conflicts

Distributed Version Control Systems



- Every checkout is really a full backup of all the data.
- Allows you to set up several types of workflows that aren't possible in centralized systems
- Other distributed systems include
 - Mercurial
 - BitKeeper
 - Darcs
 - Bazaar

DVCS Advantages

- Every clone is a backup.
- It works offline.
- It's fast.
- It handles changes well.
- Branching and merging is easy.
- Less management.

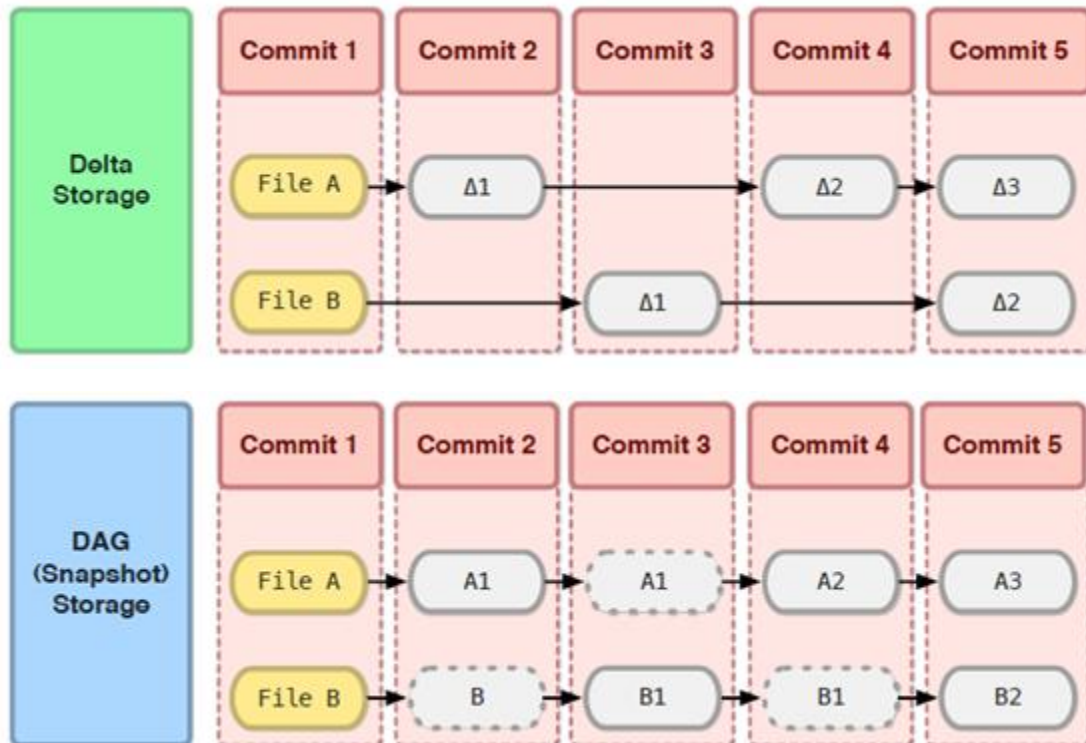
Everything is Local (Almost)

- One huge benefit is that almost all version control operations happen locally.
- Aside from sync, and then only if sync is not between two local repositories.

Create Repo	Rebase	Diff
Commit	Tag	History
Merge	Status	Bisect
Branch	Revision	Local Sync

Storage

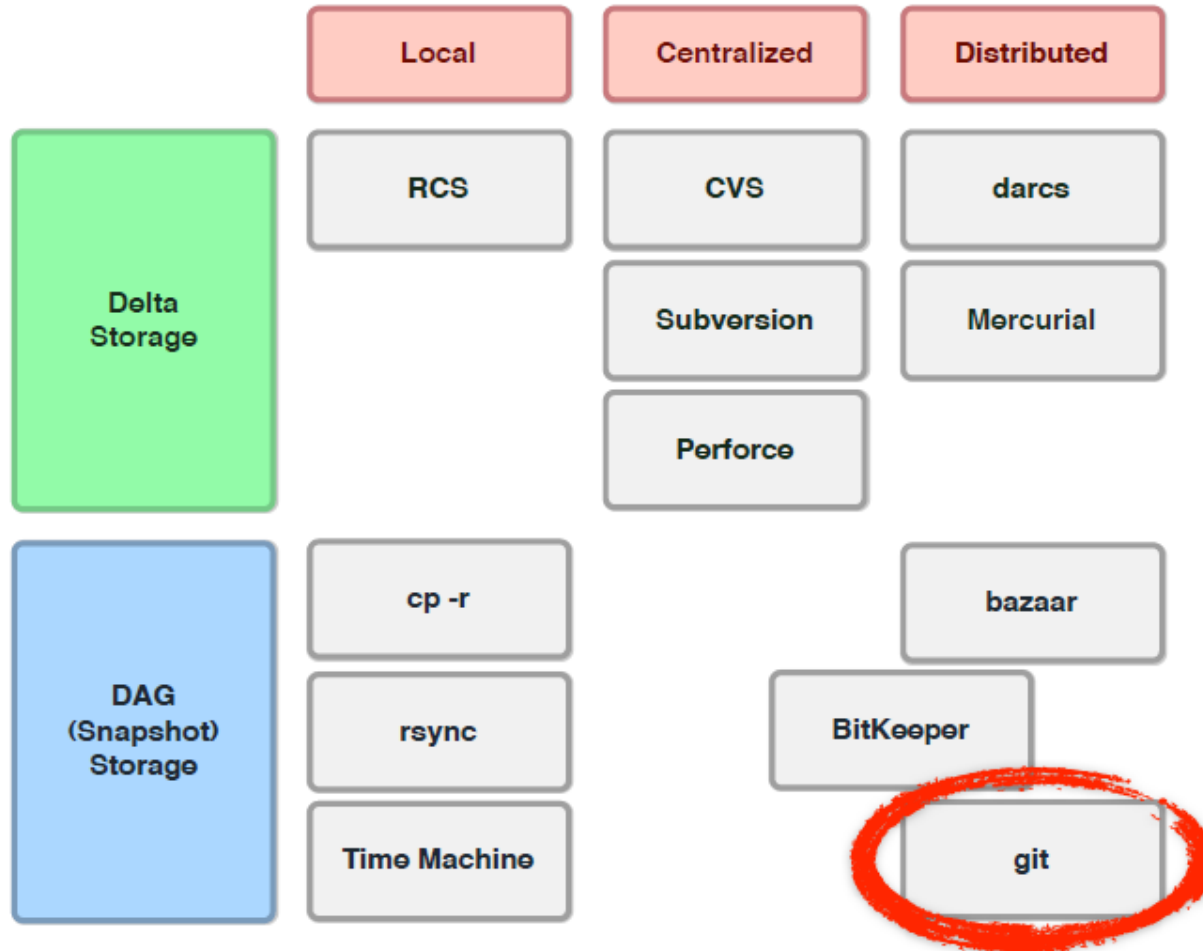
There are two ways VCS store their data



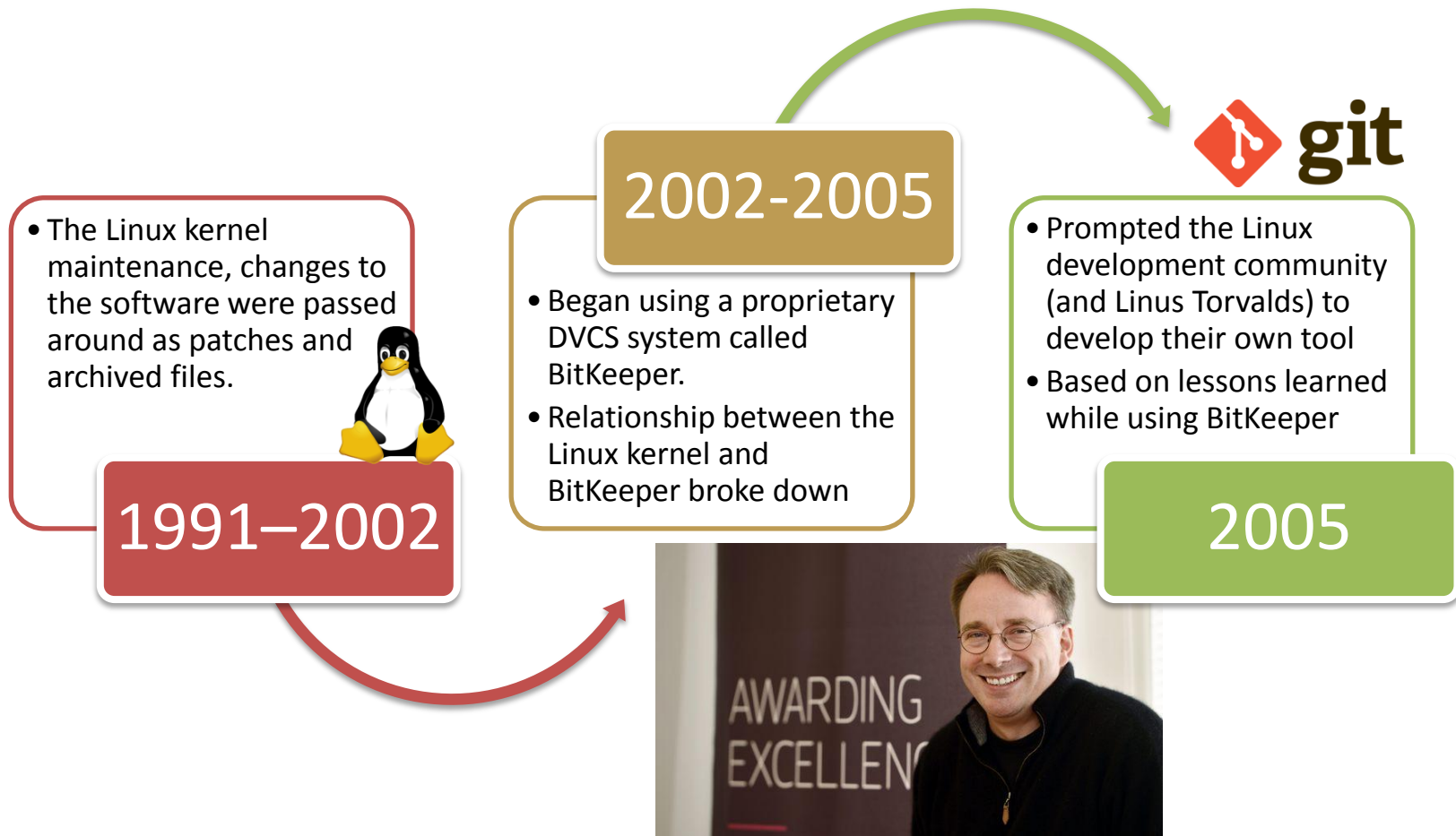
- **Delta Storage:** store data as changes to a base version of each file.
- **Snapshot Storage (a.k.a. Direct Acyclic Graph):** Each commit takes a full snapshot of your entire working directory.

Delta vs. Snapshot

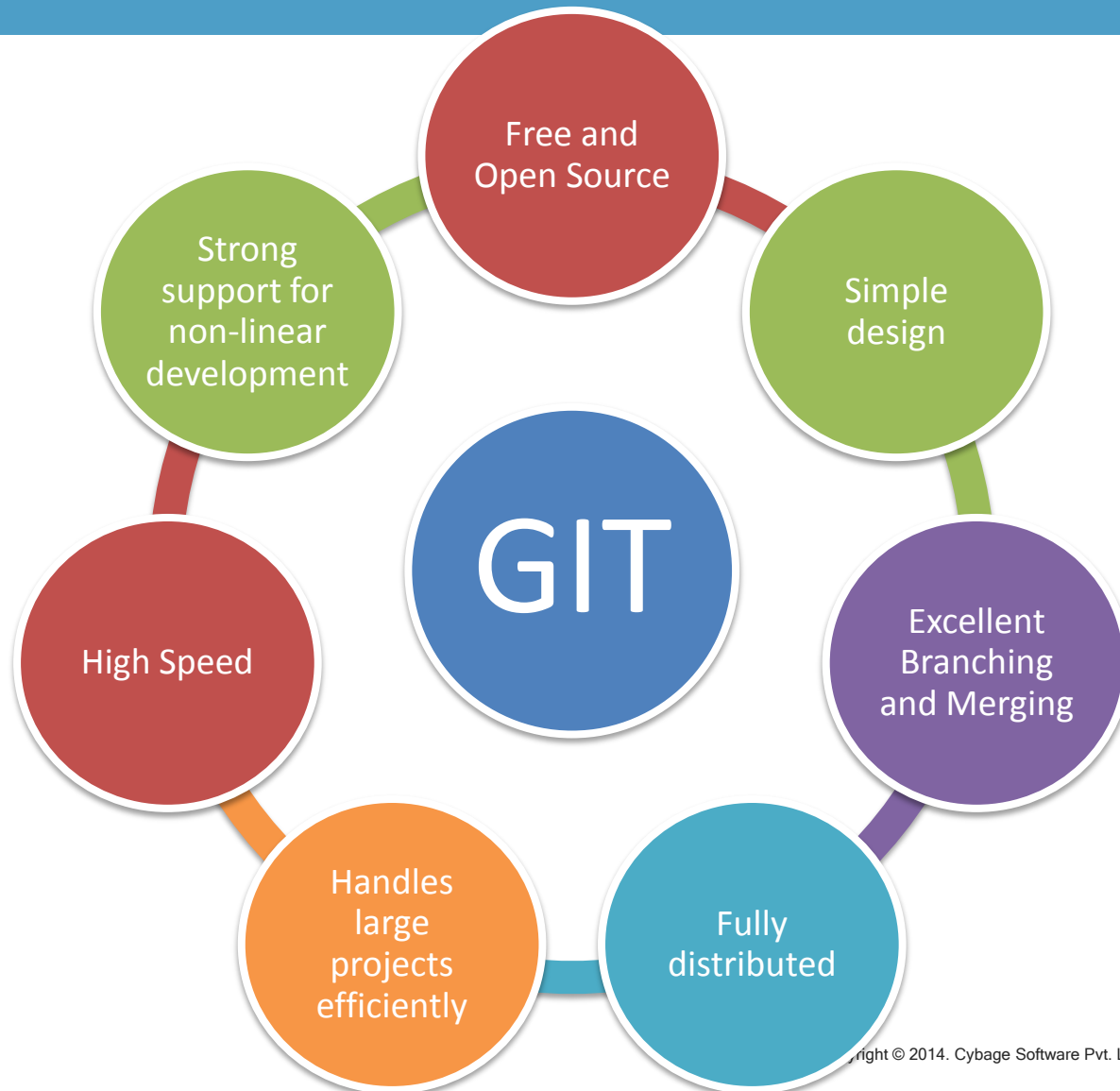
Git is a distributed VCS that uses the snapshot storage model.



History of GIT

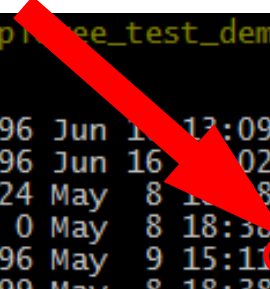


Why GIT?



Git Directory

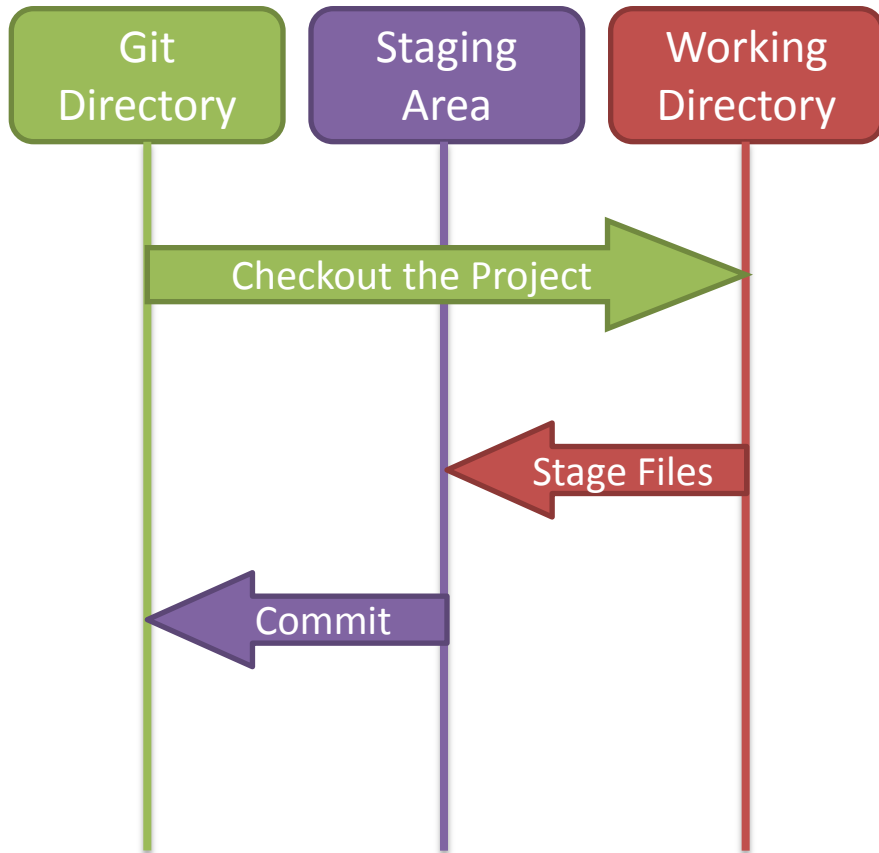
.git only in root of Working Directory(unlike Subversion)



```
prajaktaba@PRAJAKTAB-WIN8 /d/ALMDemo/employee_test_demo (master)
$ ls -la
total 12
drwxr-xr-x 10 prajakta Administ 4096 Jun 1 17:09 .
drwxr-xr-x 1 prajakta Administ 4096 Jun 16 02:02 ..
-rw-r--r-- 1 prajakta Administ 1024 May 8 18:38 .classpath
drwxr-xr-x 4 prajakta Administ 0 May 8 18:38 .clover
drwxr-xr-x 12 prajakta Administ 4096 May 9 15:11 .git
-rw-r--r-- 1 prajakta Administ 1599 May 8 18:38 .project
drwxr-xr-x 11 prajakta Administ 4096 May 8 18:38 .settings
drwxr-xr-x 3 prajakta Administ 0 May 8 18:38 META-INF
drwxr-xr-x 4 prajakta Administ 0 May 8 18:38 src
drwxr-xr-x 11 prajakta Administ 4096 May 8 18:38 target

prajaktaba@PRAJAKTAB-WIN8 /d/ALMDemo/employee_test_demo (master)
$ ls -ls .git/
total 33
 1 -rw-r--r-- 1 prajakta Administ 23 May 8 18:38 HEAD
 1 -rw-r--r-- 1 prajakta Administ 414 May 8 18:45 config
 1 -rw-r--r-- 1 prajakta Administ 73 May 8 18:37 description
 2 drwxr-xr-x 12 prajakta Administ 4096 May 8 18:37 hooks
27 -rw-r--r-- 1 prajakta Administ 53626 May 9 15:11 index
 0 drwxr-xr-x 3 prajakta Administ 0 May 8 18:37 info
 0 drwxr-xr-x 4 prajakta Administ 0 May 8 18:38 logs
 2 drwxr-xr-x 12 prajakta Administ 4096 May 9 15:11 objects
 1 -rw-r--r-- 1 prajakta Administ 329 May 8 18:38 packed-refs
 0 drwxr-xr-x 5 prajakta Administ 0 May 8 18:38 refs
```

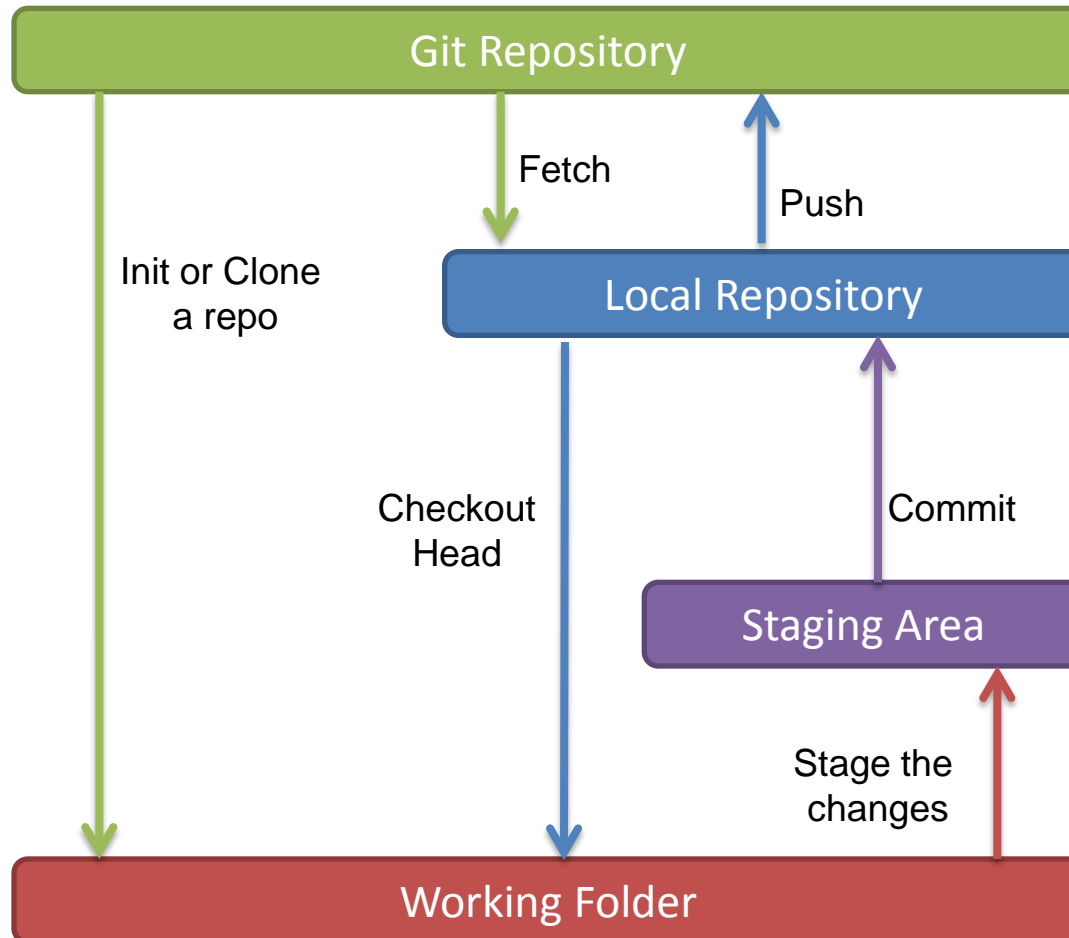
Getting Started



This leads us to the three main sections of a Git project

- **Git Directory:** is where Git stores the metadata and object database for your project.
- **Staging Area:** stores information about what will go into your next commit. Also called 'Index'.
- **Working directory:** is a single checkout of one version of the project

Basic Workflow



Git Branching and Merging Features

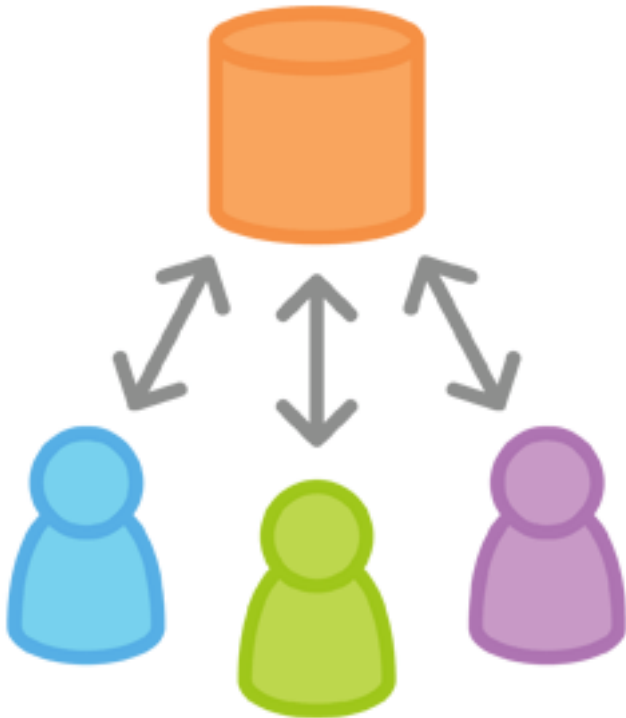
Some people refer to the branching model in Git as its **"killer feature"**

- Lightweight
- Nearly instantaneous
- Switching back and forth between branches generally just as fast
- Git encourages a workflow that branches and merges often
- Non-linear development(Parallelize)
- Easy to handle Long Running Topics
- Isolate Experiments
- Hot Fix

Git Workflows

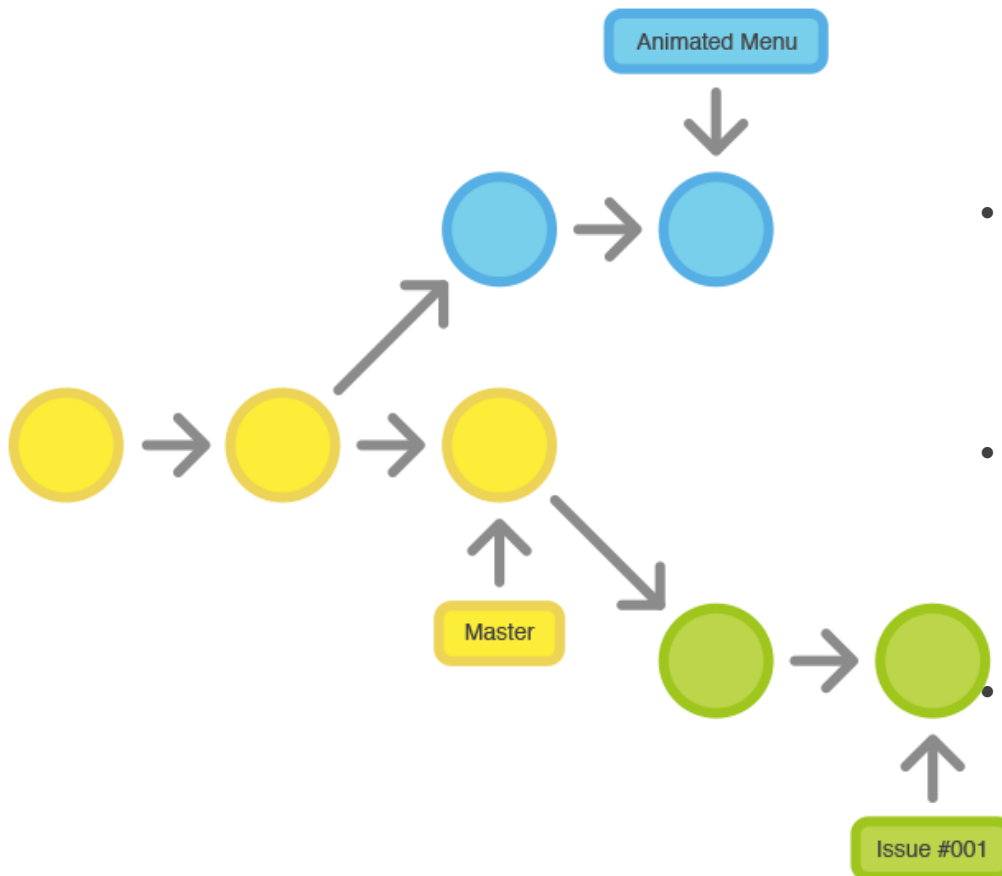
- Git workflows are designed to be guidelines rather than concrete rules.
- You can mix and match aspects from different workflows to suit your individual needs.
- Major workflows are
 - ✓ Centralized Workflow
 - ✓ Feature Branch Workflow
 - ✓ Gitflow Workflow
 - ✓ Forking Workflow

Centralized Workflow



- Your team can develop projects in the exact same way as they do with Subversion.
- Gives every developer their own local copy of the entire project.
- Gives you access to Git's robust branching and merging model.
- Comfort of Subversion, advantages of Git!!!

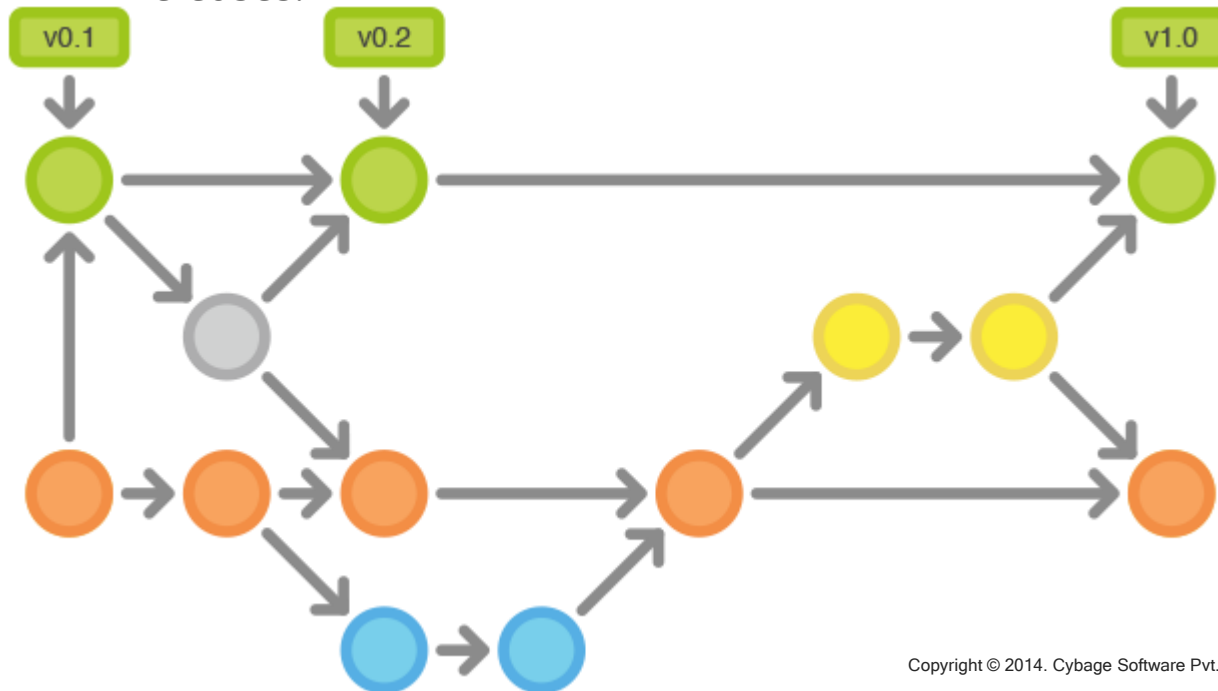
Feature Branch Workflow



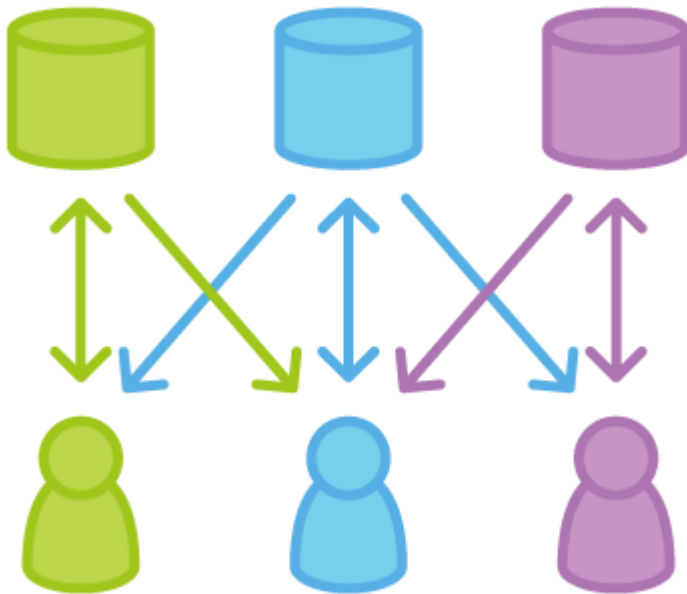
- To encourage collaboration and streamline communication between developers.
- All feature development should take place in a dedicated branch instead of the master branch.
- Easy for multiple developers to work on a particular feature without disturbing the main codebase
- The master branch will never contain broken code, which is a huge advantage for continuous integration environments.

Gitflow Workflow

- Defines a strict branching model designed around the project release.
- Provides a robust framework for managing larger projects.
- Assigns very specific roles to different branches and defines how and when they should interact.
- Uses individual branches for preparing, maintaining, and recording releases.



Forking Workflow



- Fundamentally different than the other workflows
- Gives every developer a server-side repository
- Each contributor has two Git repositories: a private local and a public server-side
- The result is a distributed workflow that provides a flexible way for large, organic teams (including untrusted third-parties) to collaborate securely.
- This also makes it an ideal workflow for open source projects.

Git Advanced Features

Rebase

- Rebasing is the process of moving a branch to a new base commit.

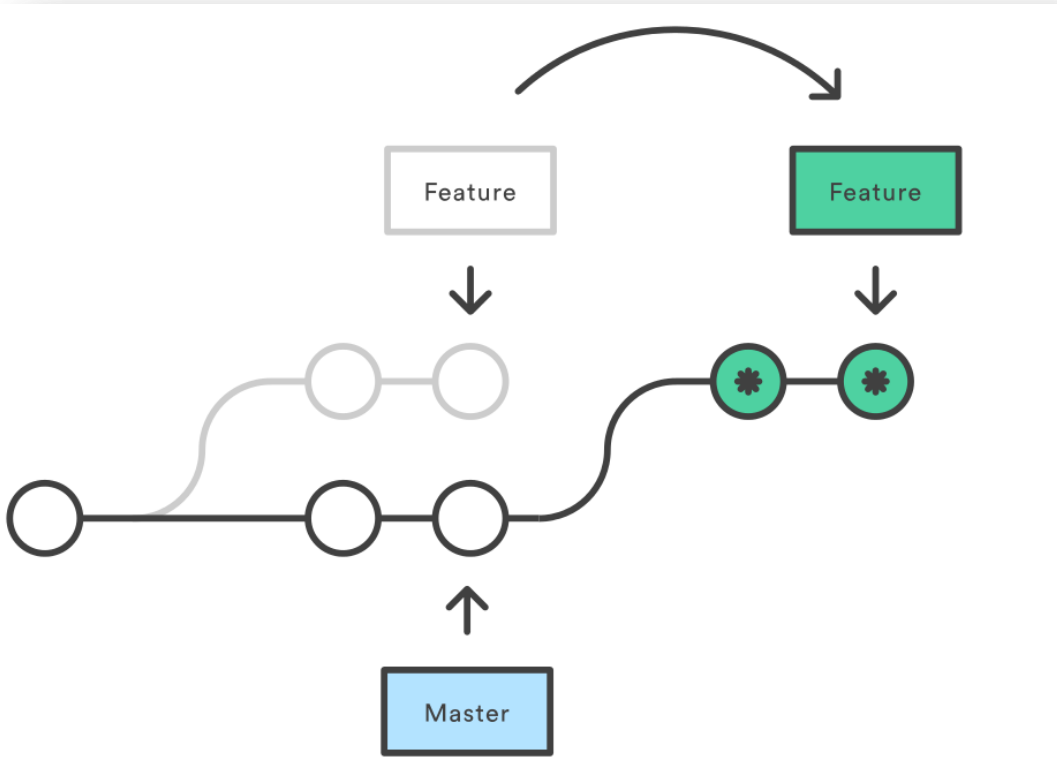
Hooks

- Git hooks are scripts that run automatically every time a particular event occurs in a Git repository.

Forks

- Forking a repository allows you to freely experiment with changes without affecting the original project.

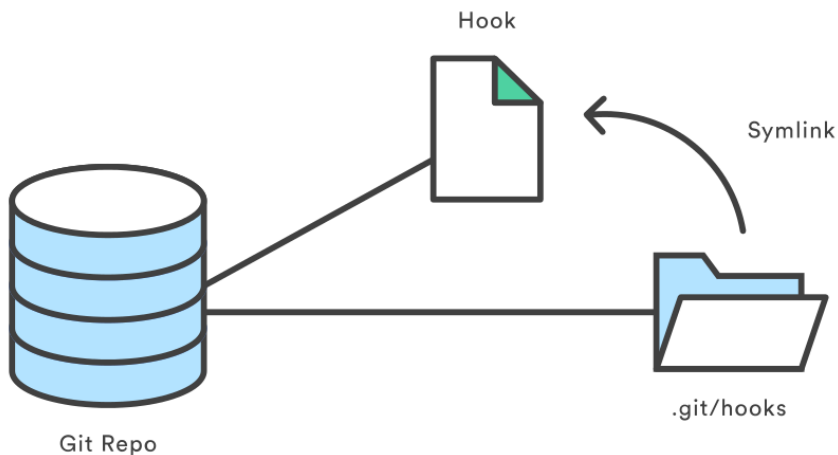
Rebase



- From a content perspective, rebasing really is just moving a branch from one commit to another.
- But internally, Git accomplishes this by creating new commits and applying them to the specified base—it's literally rewriting your project history.
- It's very important to understand that, even though the branch looks the same, it's composed of entirely new commits.

Hooks

Maintaining a hook using a symlink to version-controlled script



- They let you customize Git's internal behavior and trigger customizable actions at key points in the development life cycle.
- Common use cases for Git hooks include encouraging a commit policy, altering the project environment depending on the state of the repository, and implementing continuous integration workflows.
- But, since scripts are infinitely customizable, you can use Git hooks to automate or optimize virtually any aspect of your development workflow.

Types of Hooks

Local hooks

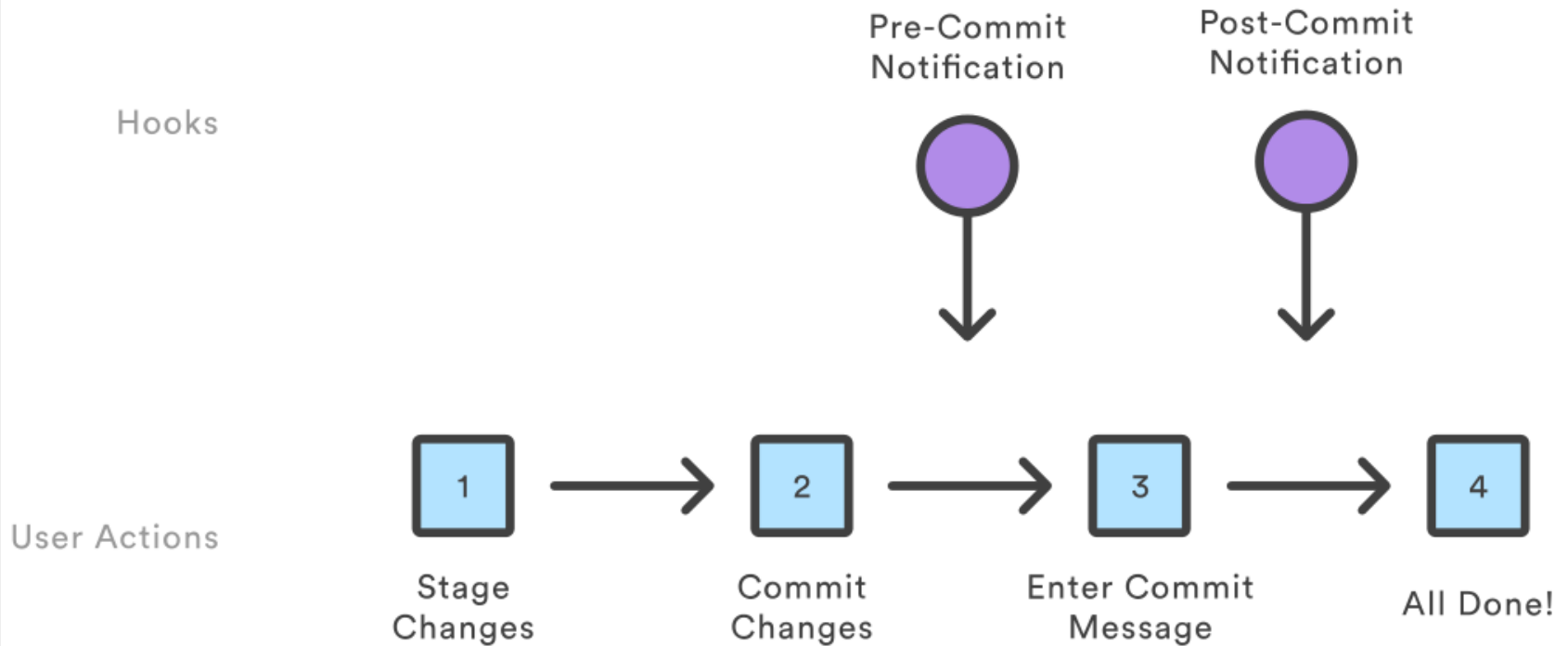
- pre-commit
- prepare-commit-msg
- commit-msg
- post-commit
- post-checkout
- pre-rebase

Server-side Hooks

- pre-receive
- update
- post-receive

How Hooks work

Hooks executing during the commit creation process



Forks

- **In recent DVCS terminology a fork is a remote, server-side copy of a repository, distinct from the original.**
- A clone is not a fork; a clone is a local copy of some remote repository.
- Another area where forks are valuable is in the interaction with third parties, contractors and freelances.
- By providing forks as the only access point for contractors to your repositories one can gain several benefits:
 - ✓ Keep your main repository clean and restricted.
 - ✓ Integrate third party work after review at scheduled times.
 - ✓ Retain the common git collaboration process.

Any Questions?





Thank You!