GIT - Version Control Systems

Ganesh Palnitkar

Agenda

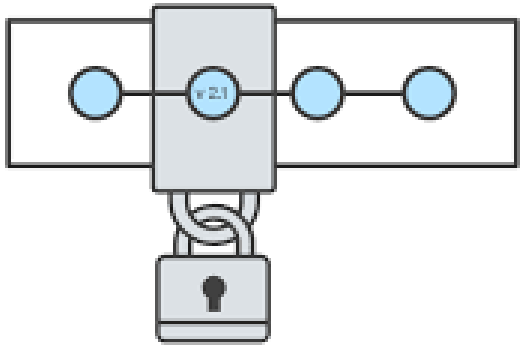
In this session, you will learn about:

- Importance of Version Control System
- Importance of Branching and Merging
- Branching and Merging Strategies
- Learning GIT – Common

About Version Control Systems

Version control software is an essential part of the every-day of the modern software team's professional practices.

- Version control software **keeps track of every modification** to the code in a special kind of database.
- Version control **protects source code** from both catastrophe and the casual degradation of human error and unintended consequences.
- **Helps teams to solve problems** like, tracking every individual change by each contributor and helping prevent concurrent work from conflict.



Conflicts if any should be solved using help of tool or manually.

Version control systems

Version control (or revision control, or source control) is all about managing multiple versions of documents, programs, web sites, etc.

Almost all “real” projects use some kind of version control

Essential for team projects, but also very useful for individual projects

Some well-known version control systems are CVS, Subversion, Mercurial, and Git

CVS and Subversion use a “central” repository; users “check out” files, work on them, and “check them in”

Mercurial and Git treat all repositories as equal

Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion

What are Version Control Systems?

Version control software is an essential part of the every-day of the modern software team's professional practices.

Advantages

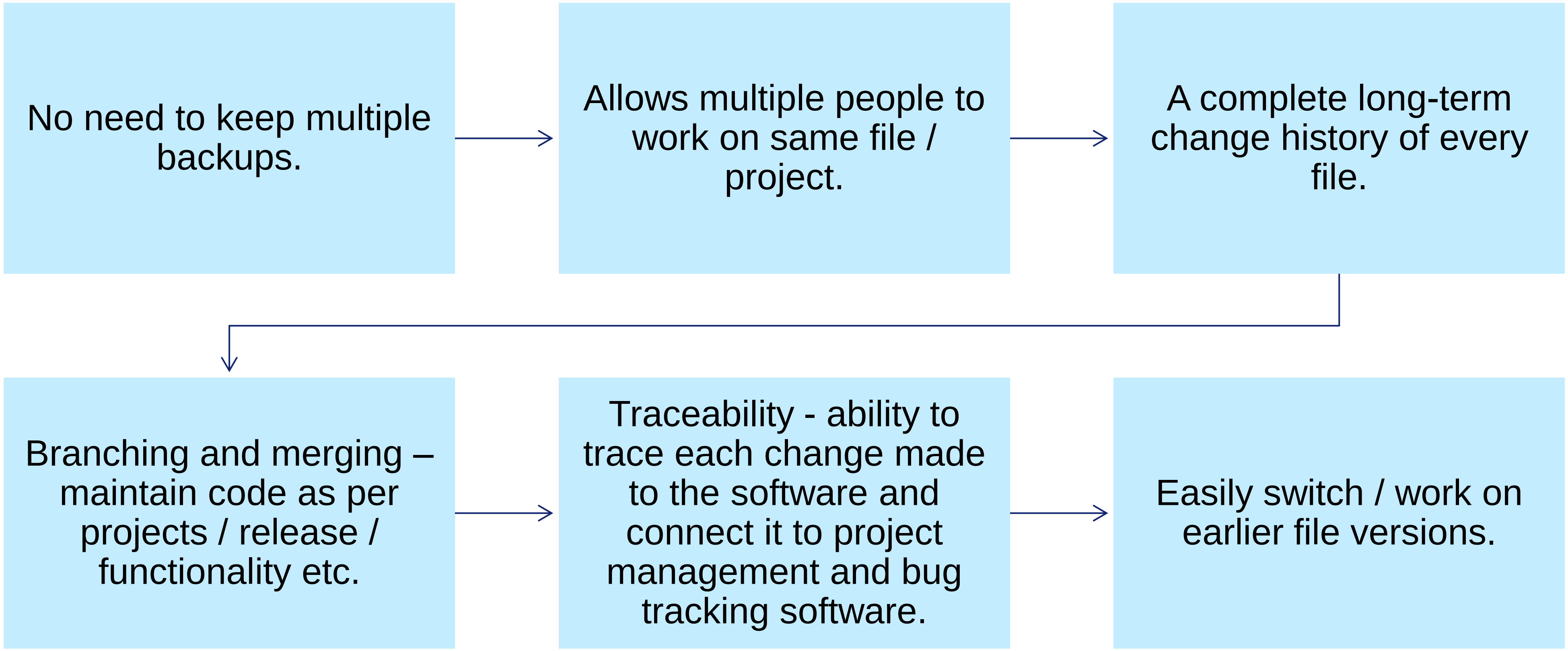
Version control software **keeps track of every modification** to the code in a special kind of database.

Version control **protects source code** from both catastrophe and the casual degradation of human error and unintended consequences.

Helps teams to solve problems like, tracking every individual change by each contributor and helping prevent concurrent work from conflict.

Conflicts if any should be solved using help of tool or manually.

Benefits of Version Control Systems



No need to keep multiple backups.

Allows multiple people to work on same file / project.

A complete long-term change history of every file.

Branching and merging – maintain code as per projects / release / functionality etc.

Traceability - ability to trace each change made to the software and connect it to project management and bug tracking software.

Easily switch / work on earlier file versions.

Git History

Came out of Linux development community

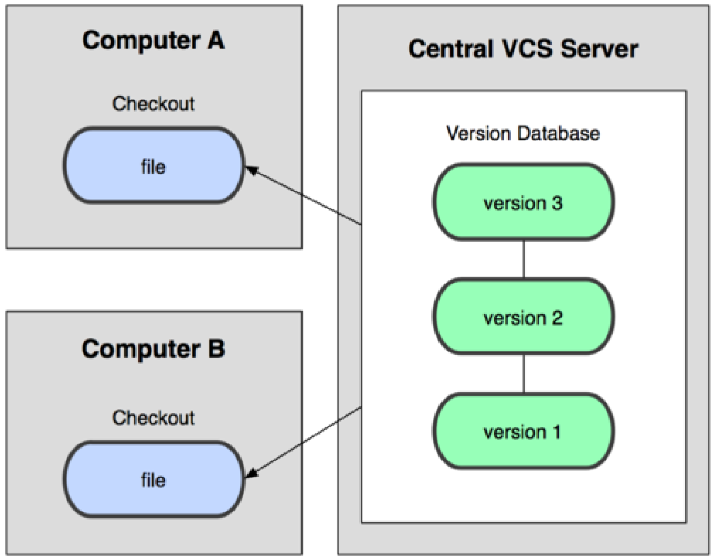
Linus Torvalds, 2005

Initial goals:

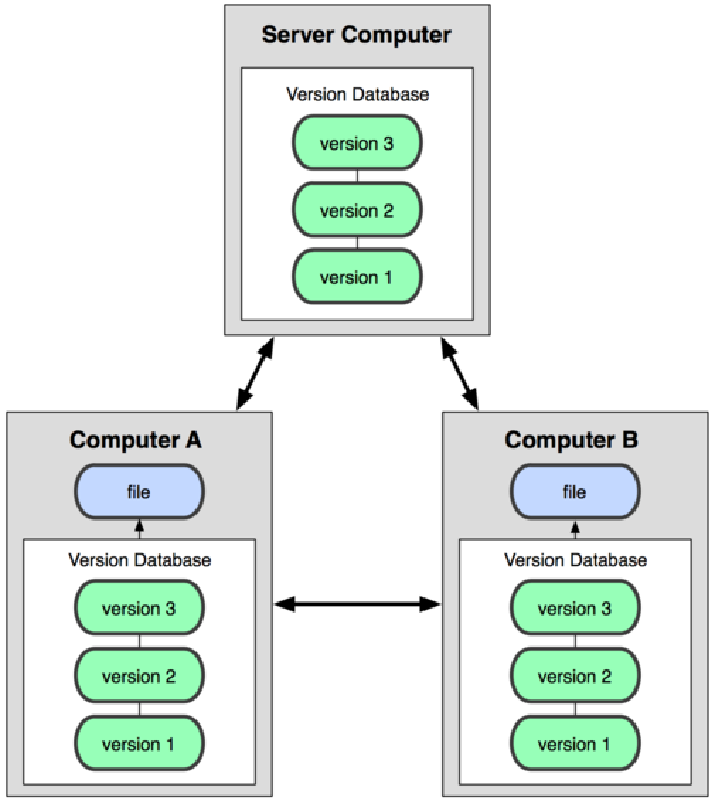
- Speed
- Support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like Linux efficiently

Git uses a distributed model

Centralized Model



Distributed Model



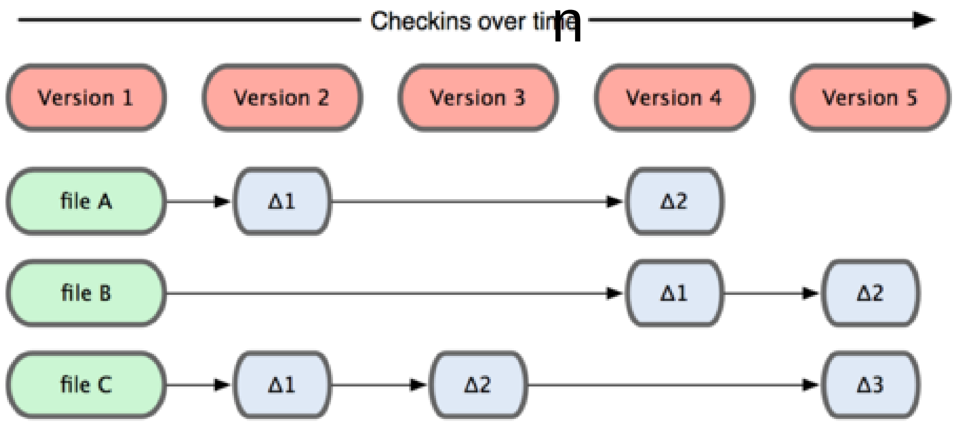
(CVS, Subversion, Perforce)

(Git, Mercurial)

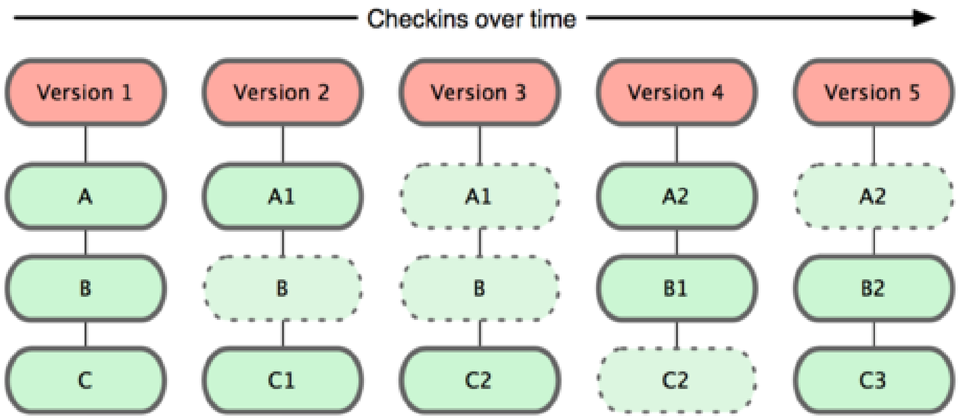
Result: Many operations are local

Git takes snapshots

Subversio
n



Git



Git uses checksums

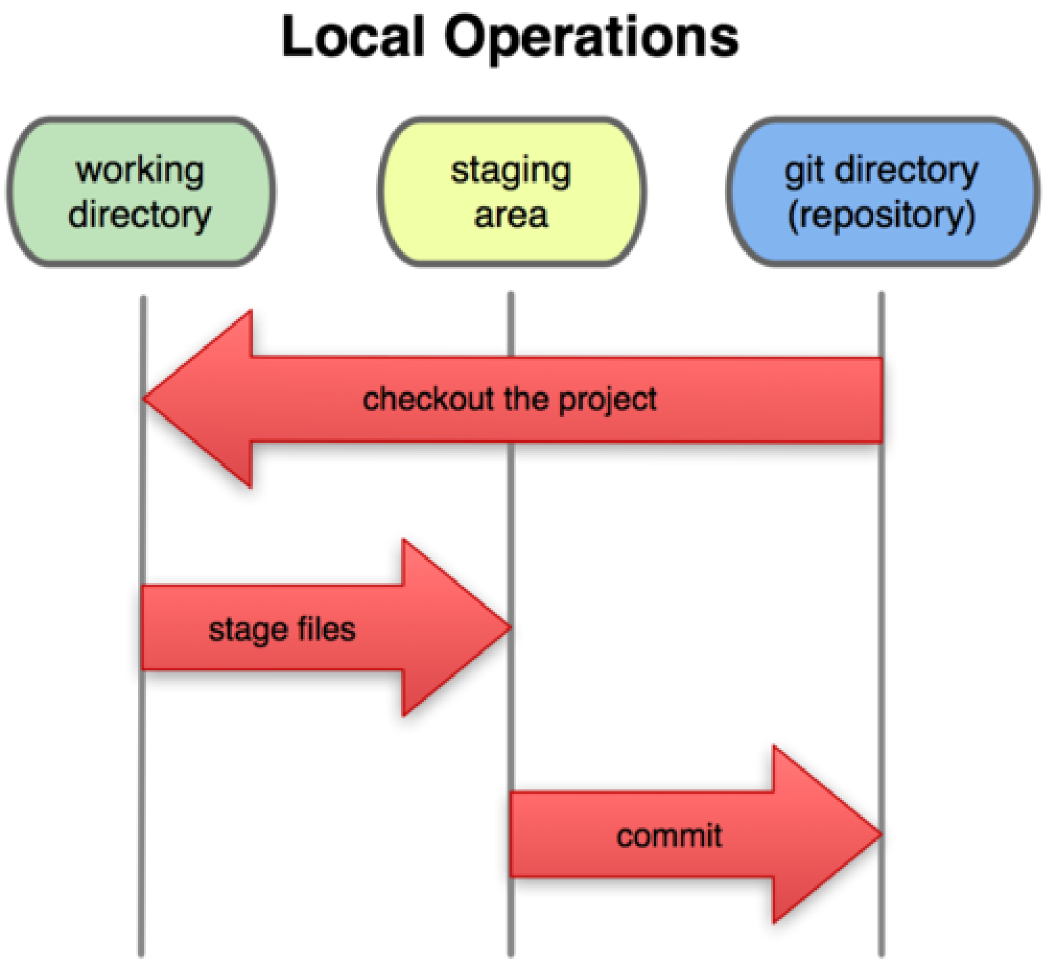
In Subversion each modification to the **central** repo incremented the version # of the overall repo.

How will this numbering scheme work **when each user has their own copy of the repo**, and commits changes to their local copy of the repo before pushing to the central server?????

Instead, Git generates a unique SHA-1 hash – 40 character string of hex digits, for every commit. Refer to commits by this ID rather than a version number. Often we only see the first 7 characters:

```
1677b2d Edited first line of readme
258efa7 Added line to readme
0e52da7 Initial commit
```

A Local Git project has three areas



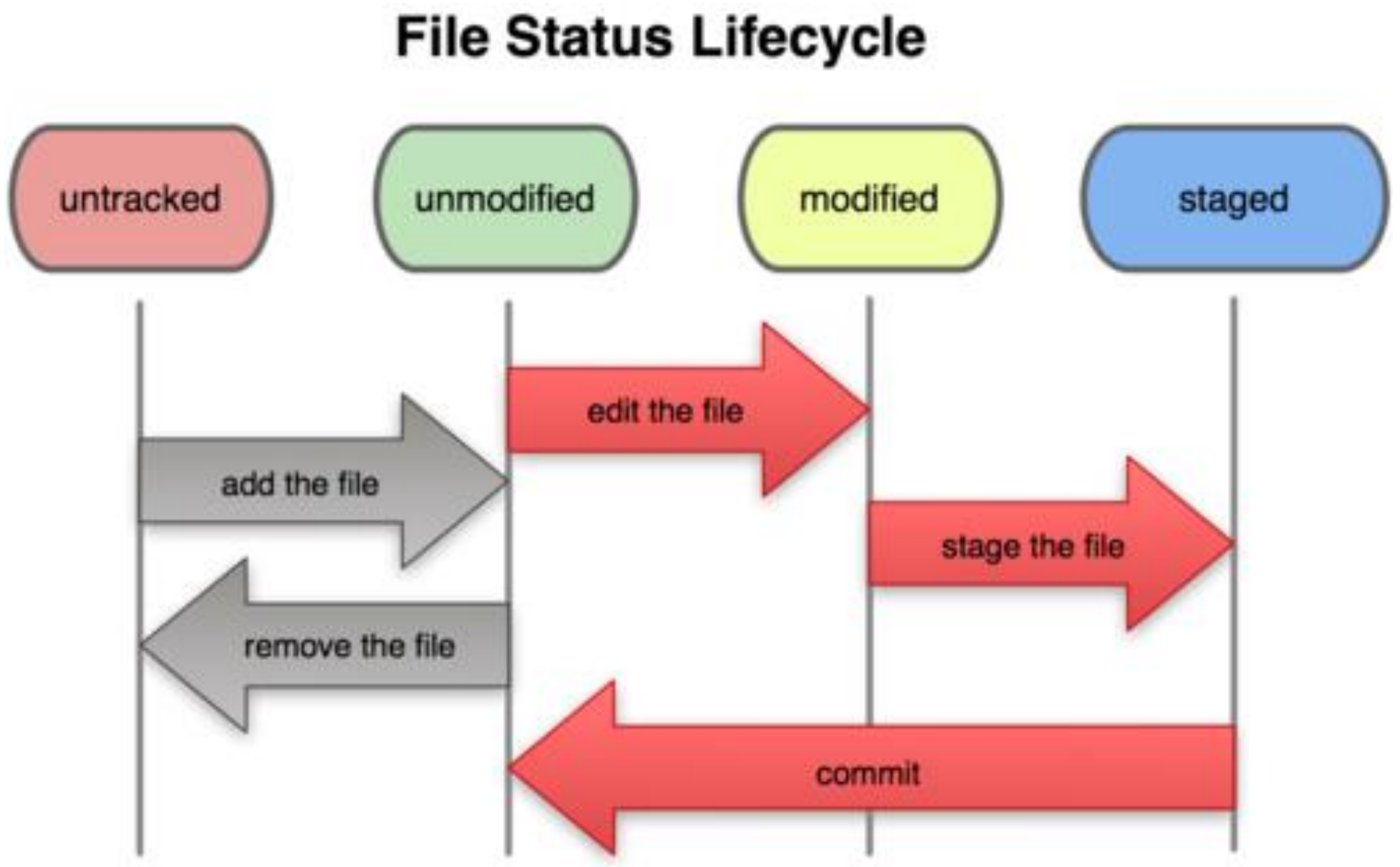
Unmodified/modified Files

Staged Files

Committed Files

Note: working directory sometimes called the “working tree”, staging area sometimes called the “index”.

Git file lifecycle



Basic Workflow

Basic Git workflow:

1. **Modify** files in your working directory.
2. **Stage** files, adding snapshots of them to your staging area.
3. Do a **commit**, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

Notes:

▪If a particular version of a file is in the **git directory**, it's considered **committed**.

▪If it's modified but has been added to the **staging area**, it is **staged**.

▪If it was **changed** since it was checked out but has <u>not</u> been staged, it is **modified**.

GIT - DVCS

GIT has bash /command-line interface and GUI as well.

- Git is a distributed VCS. Provides extremely fast operation.
- Does not need centralized server.
- Once the entire repository is pulled on local machine, network / internet connection is not required for most of the VCS operations.

Why Git?

- Git has many advantages over earlier systems such as CVS and Subversion
  - More efficient, better workflow, etc.
  - See the literature for an extensive list of reasons
  - Of course, there are always those who disagree
- Best competitor: Mercurial
  - I like Mercurial better
  - Same concepts, slightly simpler to use
  - In my (very limited) experience, the Eclipse plugin is easier to install and use
  - Much less popular than Git

Download and install Git

- There are online materials that are better than any that I could provide
- Here's the standard one: http://git-scm.com/downloads
- Here's one from StackExchange: http://stackoverflow.com/questions/315911/git-for-beginners-the-definitive-practical-guide#323764

- Note: Git is primarily a command-line tool
- I prefer GUIs over command-line tools, but…
- The GIT GUIs are more trouble than they are worth (YMMV)

Introduce yourself to Git

- Enter these lines (with appropriate changes):
  - `git config --global user.name "John Smith"`
  - `git config --global user.email jsmith@seas.upenn.edu`
- You only need to do this once

- If you want to use a different name/email address for a particular project, you can change it for just that project
  - `cd` to the project directory
  - Use the above commands, but leave out the `--global`

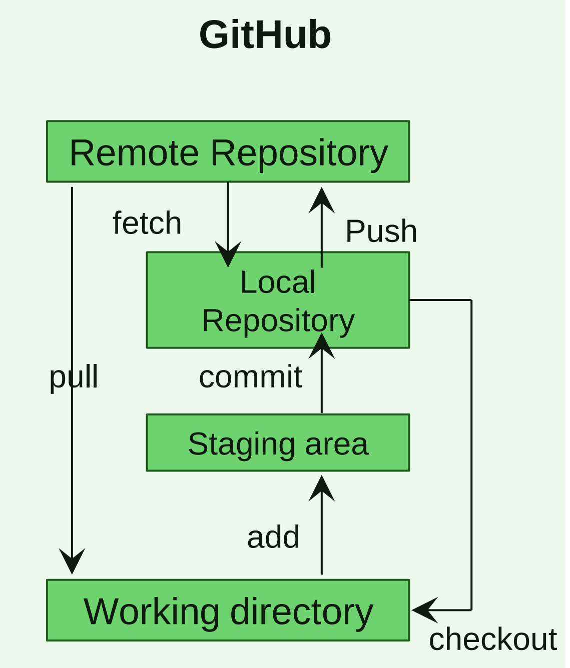
GIT - DVCS

- **Working area:** Similar to work area, development environment.
- **Staging area:** Where you store a place a snapshot before committing changes to the local repo.
- **Local repo:** Is a copy of remote repository. This is where you have all versioning of data/code/artifacts maintained.
- **Remote repository:** This can be the GitHub repository or a remote repository maintained on a server on intranet.

- GIT works on most of the OS platforms (OSx, Windows, Unix / Linux).

- On installation, the GIT config should be run to configure user name, email ID, etc.

- In order to start using the VCS, the folder that you want to version control, run the command 'git init'. This enables the VCS and tracking of file changes in the folder.



Download 'git' from, http://git-scm.com

Sprint Goal

A short statement of what the work will be focused on during the sprint

Database Application

Make the application run on SAP HANA Server in addition to SQL.

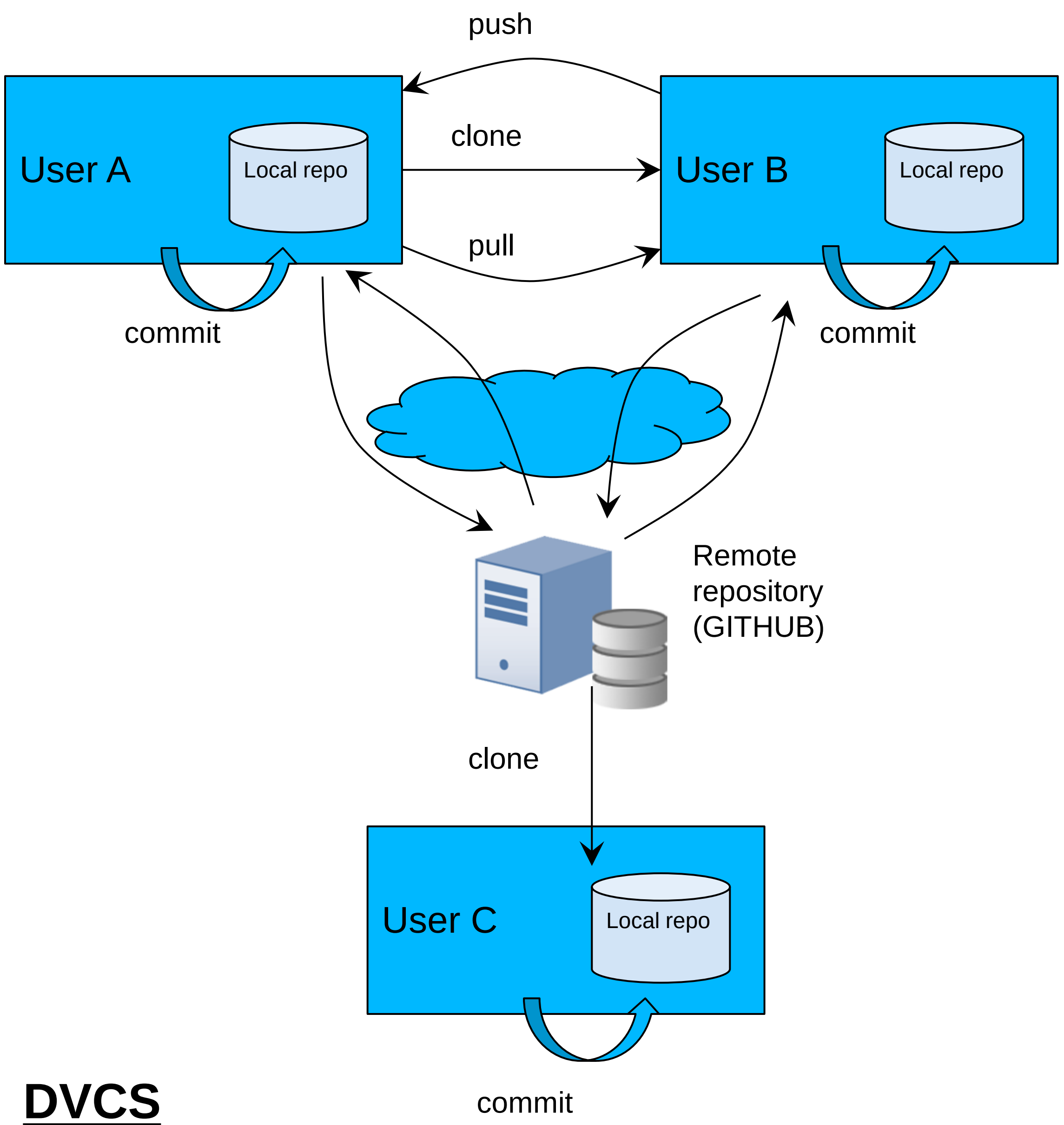
Life Sciences

Support features necessary for critical ailments relation to genes studies.

Financial Services

Support Interfaces rather than streaming real time data.

Distributed VCS & Centralized VCS



DVCS

GIT Common Commands

Below are certain commands that are used often while we use ‘GIT’.

git status

git add

git rm –cached -- to remove file added to staging area

git commit –m “comment”

git log --- use ‘shift+z’ to come out of the log

git diff -- provides info about what has changed in the file

git diff –cached   --- diff for the files in the staging area.

git branch -- provides list of all branches

git branch <branch name> -- create a branch

git remote add <name> <url> -- to add remote repo.

git checkout <branch name> -- switch to mentioned branch

git clone <https://remote repo> -- clone from remote repo.

git pull <repo name> <branch name>

Always perform a ‘Pull’ action before ‘pushing’ the code to remote repo.

Git commands

command	description
`git clone` ***`url [dir]`***	copy a git repository so you can add to it
`git add` ***`files`***	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`	

Create and fill a repository

1. `cd` to the project directory you want to use
2. Type in `git init`
   - This creates the repository (a directory named `.git`)
   - You seldom (if ever) need to look inside this directory
3. Type in `git add .`
   - The period at the end is part of this command!
     - Period means "this directory"
   - This adds all your current files to the repository
4. Type in `git commit -m "Initial commit"`
   - You can use a different commit message, if you like

Clone a repository from elsewhere

- `git clone` *`URL`*
- `git clone` *`URL mypath`*
  - These make an exact copy of the repository at the given URL
- `git clone git://github.com/`*`rest_of_path/file.git`*
  - Github is the most popular (free) public repository
- All repositories are equal
  - But you can treat some particular repository (such as one on Github) as the "master" directory
- Typically, each team member works in his/her own repository, and "merges" with other repositories as appropriate

The repository

- Your top-level **working directory** contains everything about your project
  - The working directory probably contains many subdirectories—source code, binaries, documentation, data files, etc.
  - One of these subdirectories, named `.git`, is your repository
- At any time, you can take a "snapshot" of everything (or selected things) in your project directory, and put it in your repository
  - This "snapshot" is called a commit object
  - The commit object contains (1) a set of files, (2) references to the "parents" of the commit object, and (3) a unique "SHA1" name
  - Commit objects do *not* require huge amounts of memory
- You can work as much as you like in your working directory, but the repository isn't updated until you `commit` something

`init` and the `.git` repository

- When you said `git init` in your project directory, or when you cloned an existing project, you created a repository
  - The repository is a subdirectory named `.git` containing various files
  - The dot indicates a “hidden” directory
  - You do *not* work directly with the contents of that directory; various git commands do that for you
  - You *do* need a basic understanding of what is in the repository

Making commits

- You do your work in your project directory, as usual
- If you create new files and/or folders, they are *not tracked* by Git unless you ask it to do so
  - `git add` *`newFile1 newFolder1 newFolder2 newFile2`*
- Committing makes a "snapshot" of everything being tracked into your repository
  - A message telling what you have done is required
  - `git commit –m "Uncrevulated the conundrum bar"`
  - `git commit`
    - This version opens an editor for you the enter the message
    - To finish, save and quit the editor
- Format of the commit message
  - One line containing the complete summary
  - If more than one line, the second line must be blank

Commits and graphs

- A commit is when you tell git that a change (or addition) you have made is ready to be included in the project
- When you commit your change to git, it creates a commit object
  - A commit object represents the complete state of the project, including all the files in the project
  - The *very first* commit object has no "parents"
  - Usually, you take some commit object, make some changes, and create a new commit object; the original commit object is the parent of the new commit object
    - Hence, most commit objects have a single parent
  - You can also merge two commit objects to form a new one
    - The new commit object has two parents
- Hence, commit objects form a **directed graph**
  - Git is all about using and manipulating this graph

Working with your own repository

- A head is a reference to a commit object
- The "current head" is called HEAD (all caps)
- Usually, you will take HEAD (the current commit object), make some changes to it, and commit the changes, creating a new current commit object
  - This results in a linear graph: A → B → C → …→ HEAD

- You can also take any previous commit object, make changes to it, and commit those changes
  - This creates a branch in the graph of commit objects
- You can merge any previous commit objects
  - This joins branches in the commit graph

Commit messages

- In git, “Commits are cheap.” Do them often.
- When you commit, you must provide a one-line message stating what you have done
  - Terrible message: “Fixed a bunch of things”
  - Better message: “Corrected the calculation of median scores”
- Commit messages can be very helpful, to yourself as well as to your team members
- You can’t say much in one line, so commit often

Choose an editor

- When you "commit," git will require you to type in a commit message
- For longer commit messages, you will use an editor
- The default editor is probably `vim`
- To change the default editor:
  - `git config --global core.editor /path/to/editor`

- You may also want to turn on colors:
  - `git config --global color.ui auto`

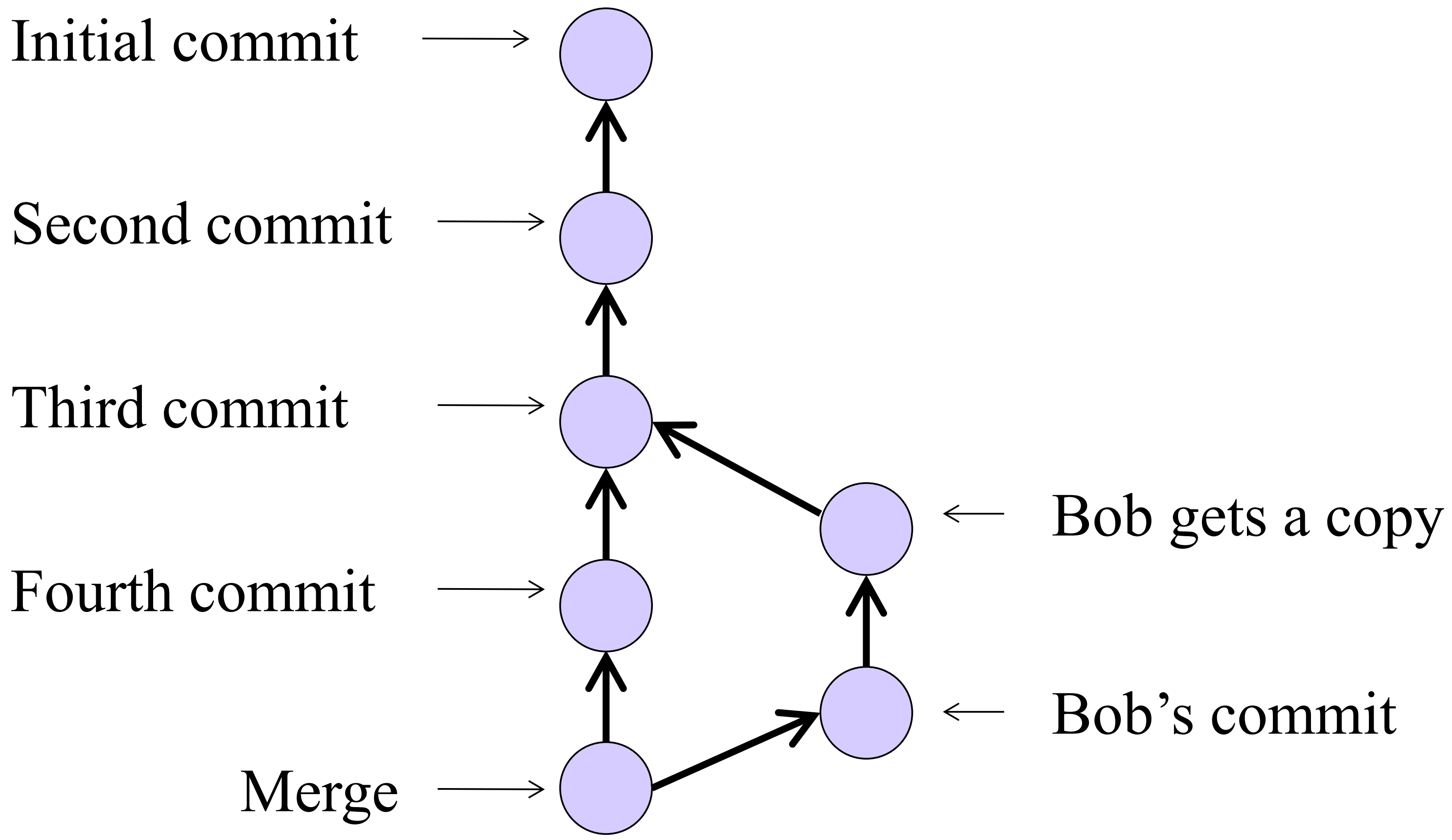
Working with others

- All repositories are equal, but it is convenient to have one central repository in the cloud
- Here's what you normally do:
  - Download the current HEAD from the central repository
  - Make your changes
  - Commit your changes to your local repository
  - Check to make sure someone else on your team hasn't updated the central repository since you got it
  - Upload your changes to the central repository
- If the central repository *has* changed since you got it:
  - It is *your* responsibility to **merge your two versions**
    - This is a strong incentive to commit and upload often!
  - Git can often do this for you, if there aren't incompatible changes

Typical workflow

- `git pull` *`remote_repository`*
  - Get changes from a remote repository and merge them into your own repository
- `git status`
  - See what Git thinks is going on
  - Use this frequently!
- Work on your files (remember to `add` any new ones)
- `git commit -m` *“What I did”*
- `git push`

Multiple versions



Aside: So what is github?

- GitHub.com is a site for online storage of Git repositories.
- Many open source projects use it, such as the Linux kernel.
- You can get free space for open source projects or you can pay for private projects.

Question: Do I have to use github to use Git?

Answer: No!

- you can use Git completely locally for your own purposes, or
- you or someone else could set up a server to share files, or
- you could share a repo with users on the same file system, such as we did for homework 9 (as long everyone has the needed file permissions).

Keeping it simple

- If you:
  - Make sure you are current with the central repository
  - Make some improvements to your code
  - Update the central repository before anyone else does
- Then you don’t have to worry about resolving conflicts or working with multiple branches
  - All the complexity in git comes from dealing with these
- Therefore:
  - Make sure you are up-to-date before starting to work
  - Commit and update the central repository frequently
- If you need help: https://help.github.com/

Activity

- Run common commands on Git
- Understand the output

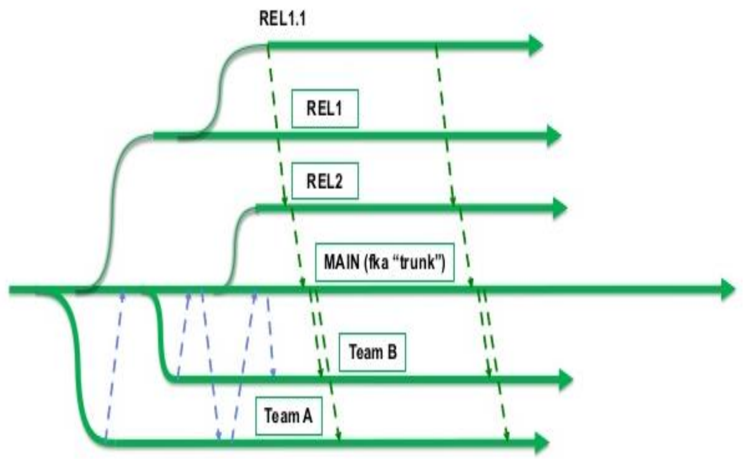
Branching and Merging Strategies

- Brach by release, “staircase model”.
- Branch by feature. Each distinct branch.
- Hotfix branch.

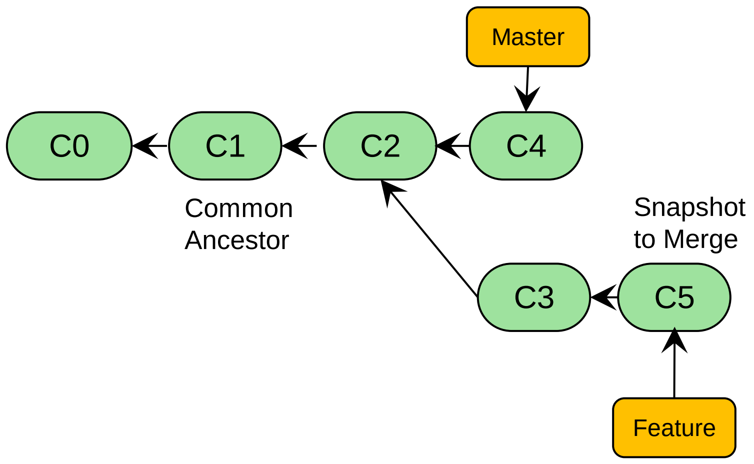
Best practices:

- Keep option of branching only if required, keep it simple.
- Incase of centralized VCS, merging should be done frequently / sooner (daily recommended)
- Use tagging as and when required.
- Delete unwanted branches.
- Identify shared components in details and in advance.

- **Merge only when your branch is compliable and stable.**

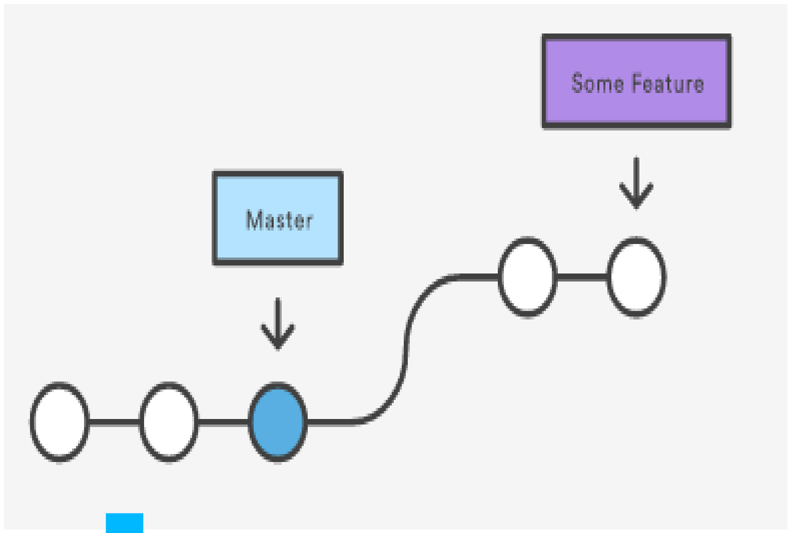


Branching and merging scenario

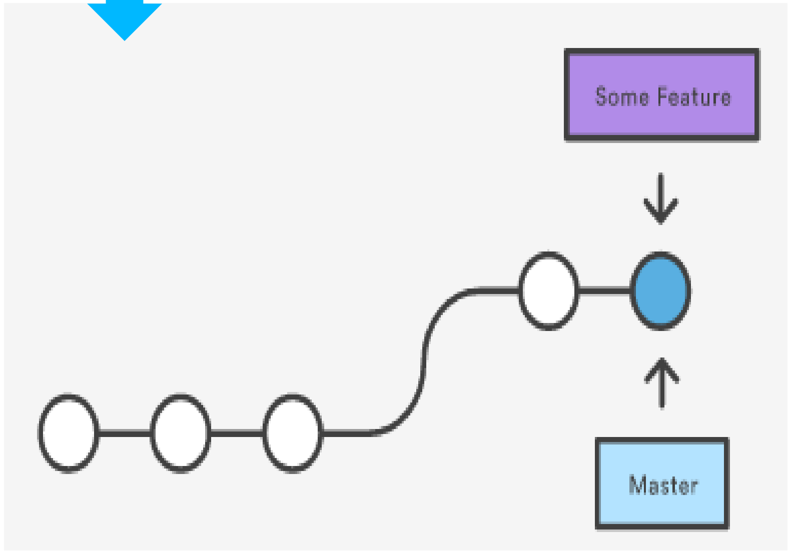


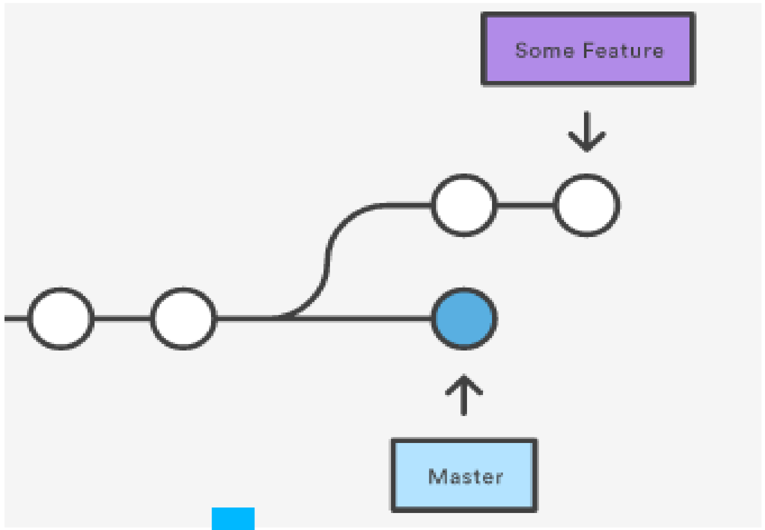
Simple Branching and merging in GIT project

GIT Merge Operations

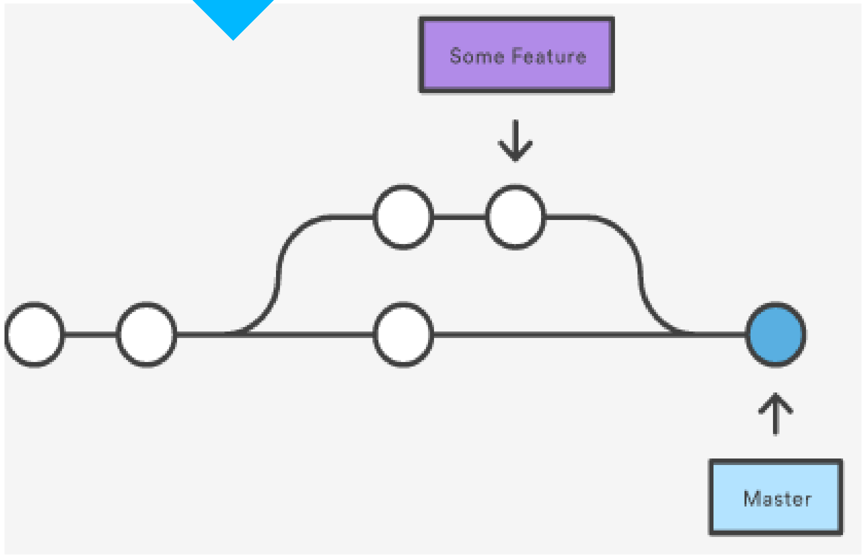


After Fast-forward Merge





3-way Merge



BitBucket - Definition

Bitbucket is Git repository management solution designed for professional teams.

Key Features of BitBucket

Gives you a central place to manage git repositories

Collaborate on your source code

Guide you through the development flow

Features of Bitbucket

Access control to restrict access to your source code.

Workflow control to enforce a project or team workflow.

Pull requests with in-line commenting for collaboration on code review.

Jira integration for full development traceability.

Full Rest API to build features custom to your workflow if they are not already available from our Marketplace

Bitbucket Cloud and Bitbucket Server

Bitbucket Cloud

- Hosted on Atlassian's servers and accessed via a URL.
- Bitbucket Cloud has an exclusive built-in continuous integration tool, Pipelines, that enables you to build, test and deploy from directly within Bitbucket.

Bitbucket Server

- Hosted on-premise, in your environment.
- Bitbucket Server does not come with a built-in testing and deployment tool, but it has strong integrations with CI / CD tool like Bamboo.

Activity

Demo of Bit Bucket – Common Operations

Questions for you

- What is the difference between pull and a check out?
- Who originally wrote git and why?
- What process should be done before each check-in?
- What types of file should NOT be included in source control?

Git Resources

At the command line: (where verb = config, add, commit, etc.)

```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

Free on-line book: http://git-scm.com/book

Git tutorial: http://schacon.github.com/git/gittutorial.html

Reference page for Git: http://gitref.org/index.html

Git website: http://git-scm.com/

Git for Computer Scientists (http://eagain.net/articles/git-for-computer-scientists/)

Thank you!