



Git – Source control
Management

Git

Agenda for today

- Objective
- Version Control and SCM
- What is Git?
- Lab 1
- Git Fundamental
- Lab 2
- Git Branching
- Lab 3
- Lab 4
- Git Collaboration
- Lab 5
- Git Workflow
- Quiz on LMS

Instructions / Notes

- Starts: **7:30am PDT/PST or 3.30pm BST or 8pm IST**
- 2 Hours session with 10 minutes Break
- Except Instructor, everyone is on mute
- Please use the Q/A Window to Ask Questions
- 5 Labs hands-on , please use laptop/desktop
- The recording & slides will be shared after session
- Ask lot of questions.
- More Info: [CloudxLab.com](https://cloudxlab.com)



Ashok Singh



Feel free to add me on linkedin

About Ashok

Founder

TejasTech

Digital Transformation Client across globe

Devops Consultant



Worked On Large Scale Computing

Computer Graduate from IIT Roorkee



Objective

Key Learning achievement



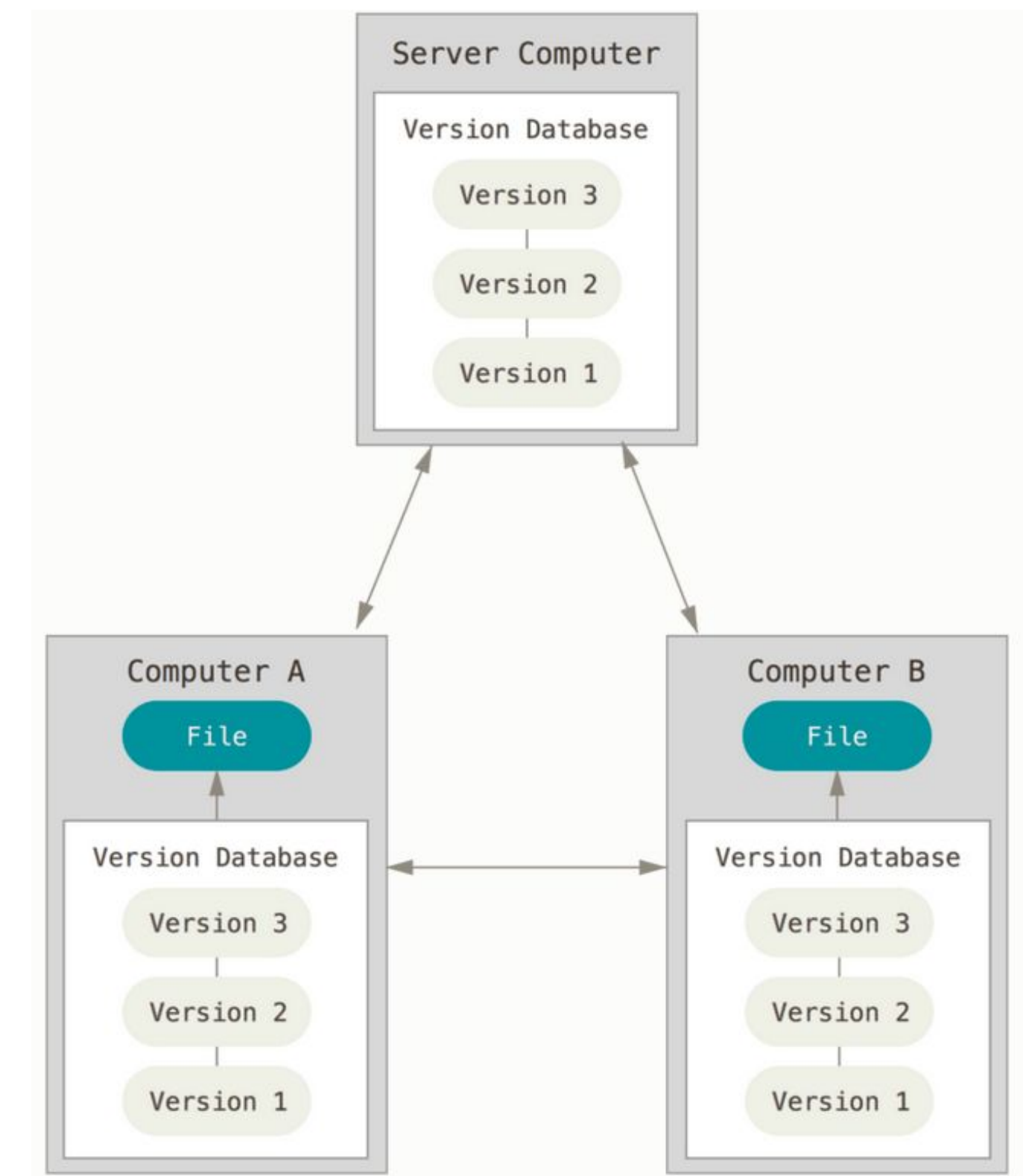
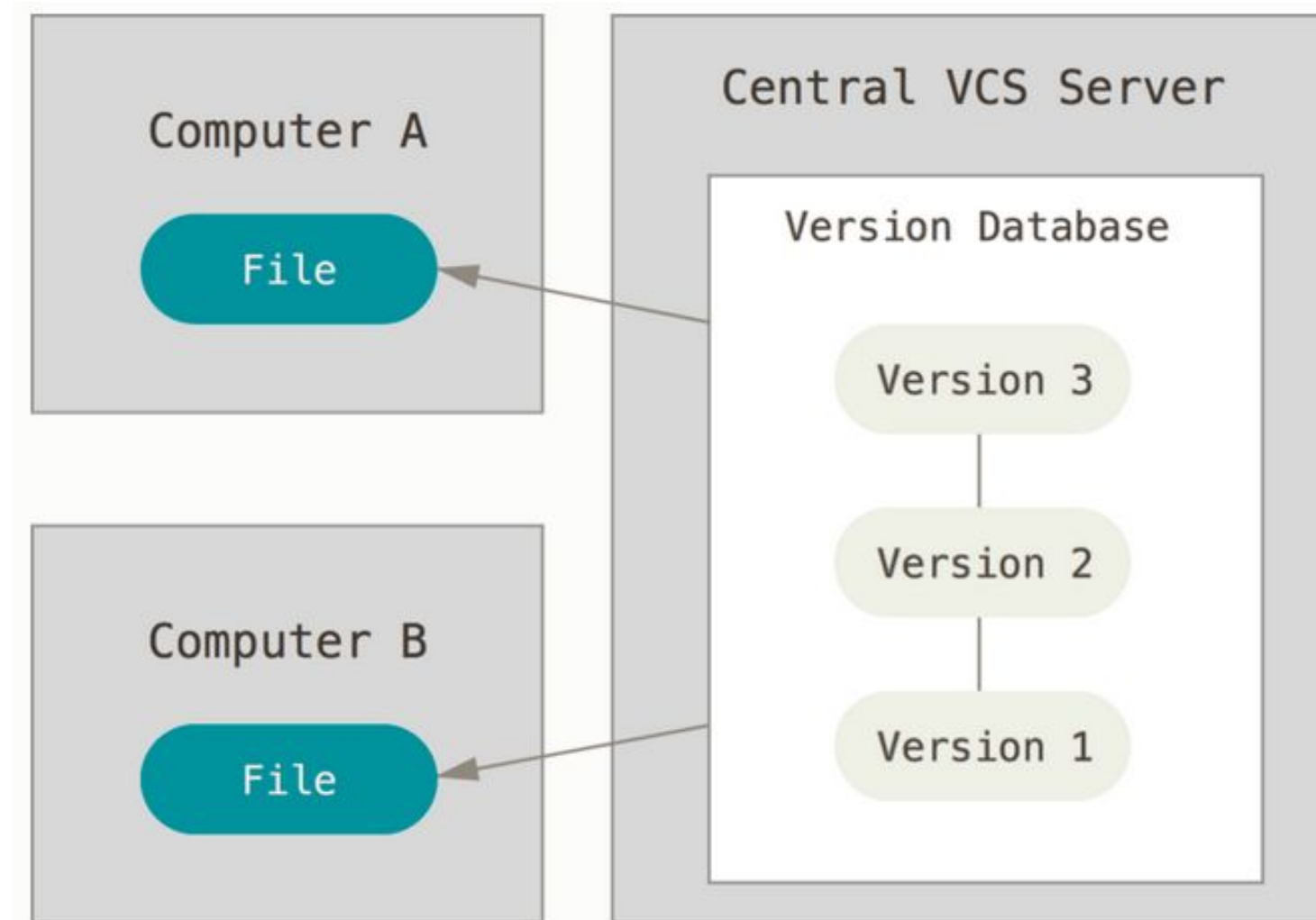
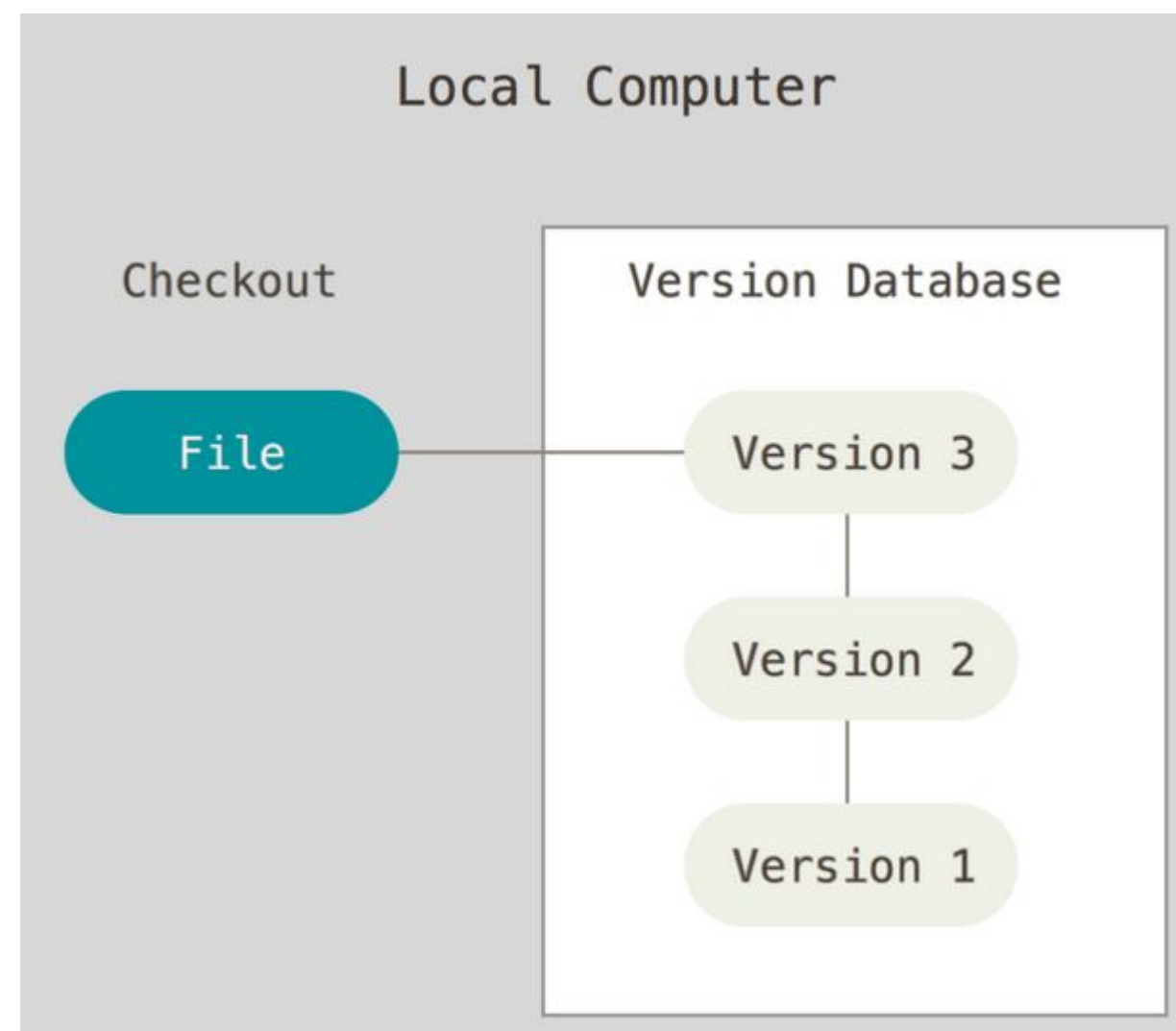
After completion of this session you would be able to :

- Install git and on any platform (Windows, Mac, Linux etc)
- Comfortably user git command for development work
- Experiment with any public available git repository on any platform
- Perform local development on open source project
- Submit developed feature for release in open source project
- Understand and implement git workflow

Version Control – What and Why ?

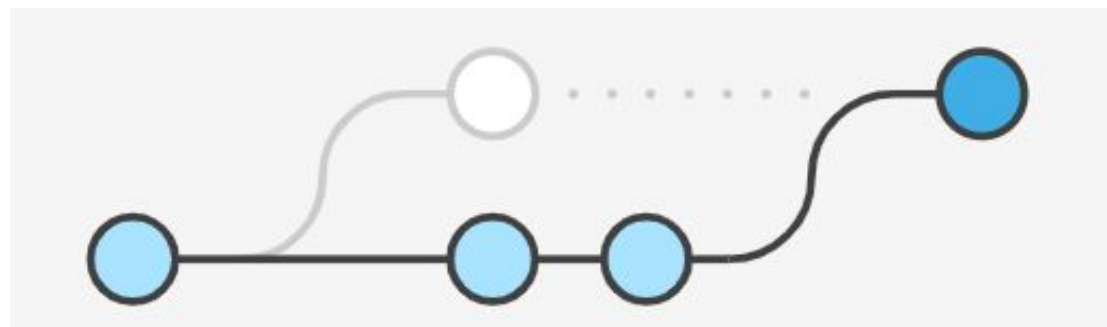
Any kind of practice that tracks and provides control over changes to software code or document.

- Local
- Centralized
- Distributed



Source Code Management (SCM)

Why SCM ?



SCM is needed when multiple contributor are working on same code base . It server the purpose to:

- Allows multiple developer work on same code for different feature .
- Undo changes
- Alleviates loads of communication between developer
- Flags conflicting changes

Version Control – Why ?

- **Backup and Restore.** Files are saved as they are edited. Wrong update can be reverted back
- **Synchronization.** Share file with developers and end user while continue to develop code
- **Short-term undo.** My build was working before , revert back to working file and compare the diff
- **Long-term undo.** Nightly builds are failing . Switch back to last working version and see what change was made that day.
- **Track Changes.** As files are committed with comments , This makes it easy to see how a file is evolving over time, and why.
- **Track Ownership.** Every commit is stored with person name
- **Sandbox :** Make micro development test changes in an controlled area and when happy commit your changes
- **Branching and merging.** clone the code locally into a separate area and modify it in controlled environemnt and later, **merge** back into the main code base.

Git – SCM

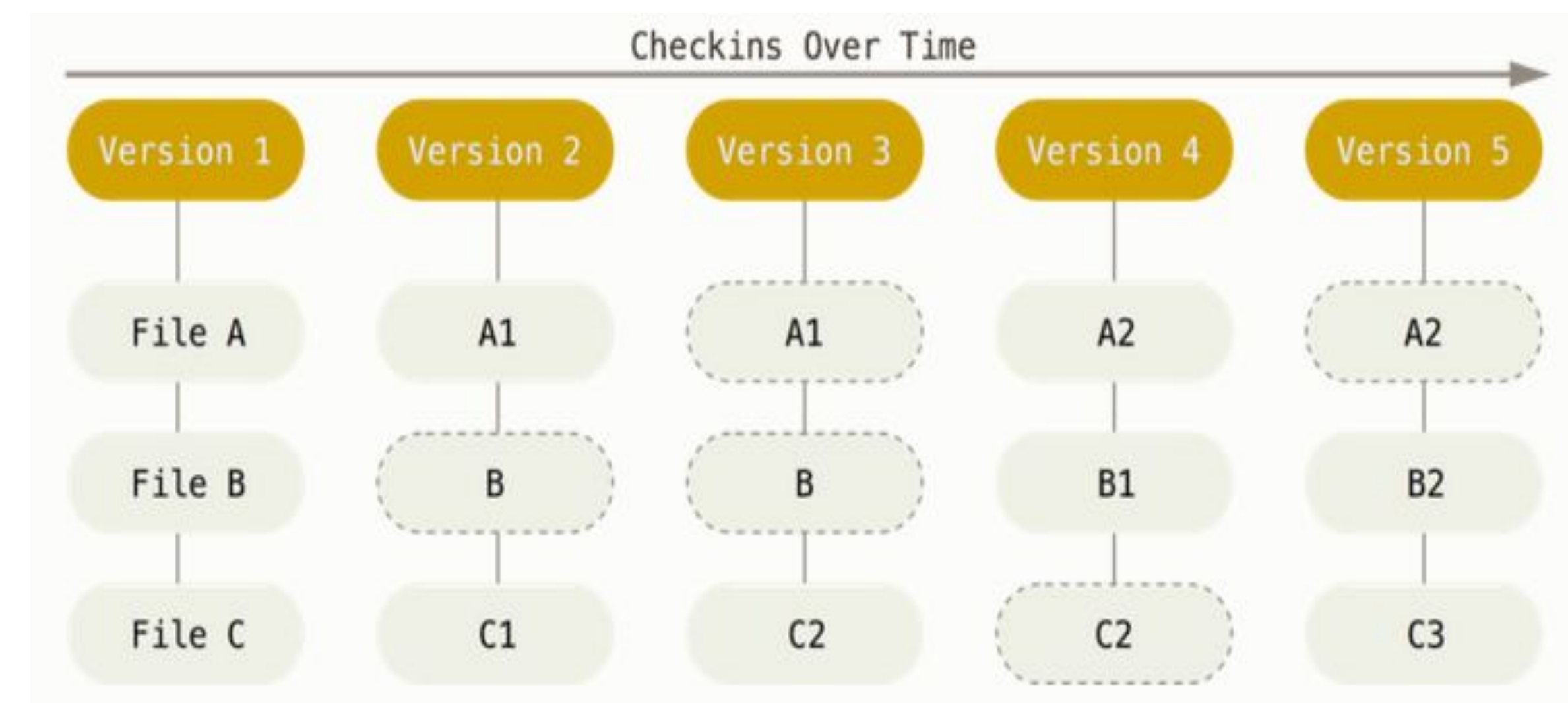
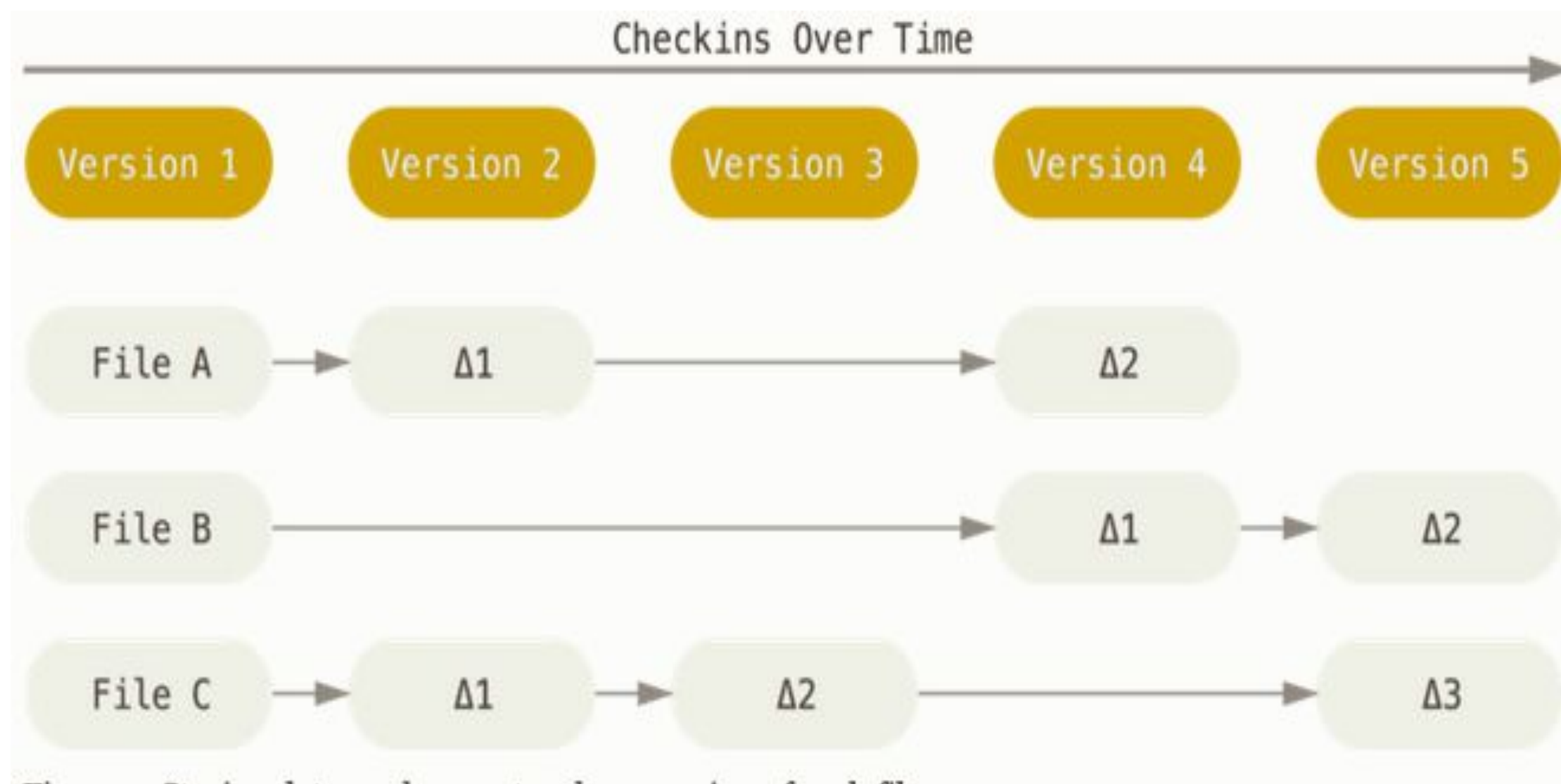


What is Git

Git is a distributed version-control system for **tracking changes in source code** during software development. It is designed for **coordinating work among programmers**, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

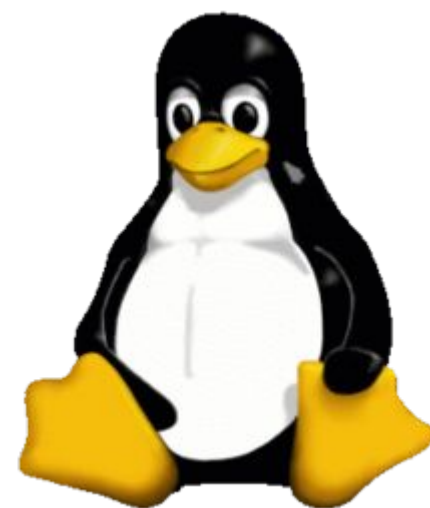
Git – SCM – Why it is different

Git thinks of its data more like a series of snapshots of a miniature filesystem. With Git, every time you commit, or save the state of your project, Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored. Git thinks about its data more like a **stream of snapshots**.



Git – History

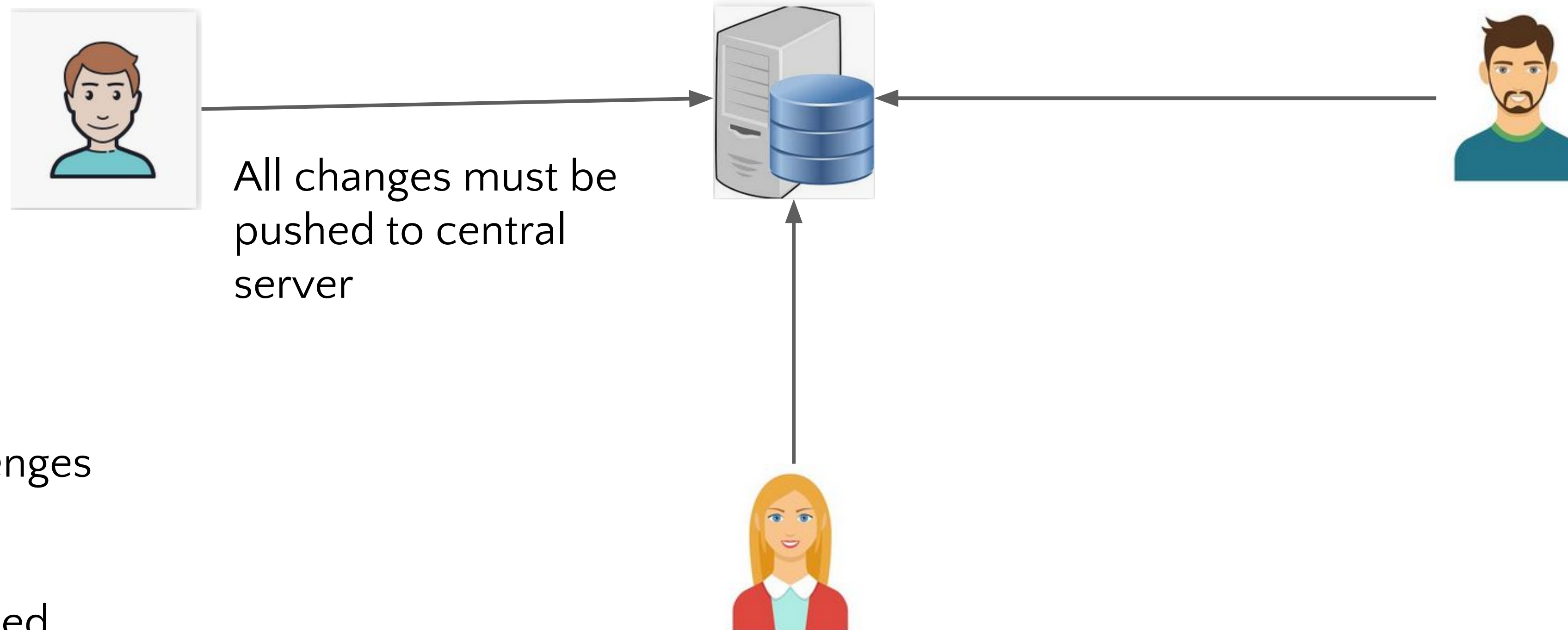
- **Until 2002**, Changes to Linux kernel were passed around as patches and archived files
- **In 2002**, Linux kernel project started using proprietary version control **BitKeeper**
- **In 2005**, the relationship between Linux community and BitKeeper broke



Git – History

- Design goals of Git
 - Speed
 - Support for non-linear development – Thousands of parallel branches
 - Fully distributed
 - Ability to handle large project like Linux Kernel efficiently
- **In 2005**, Linus Torvalds, the creator of Linux, developed Git

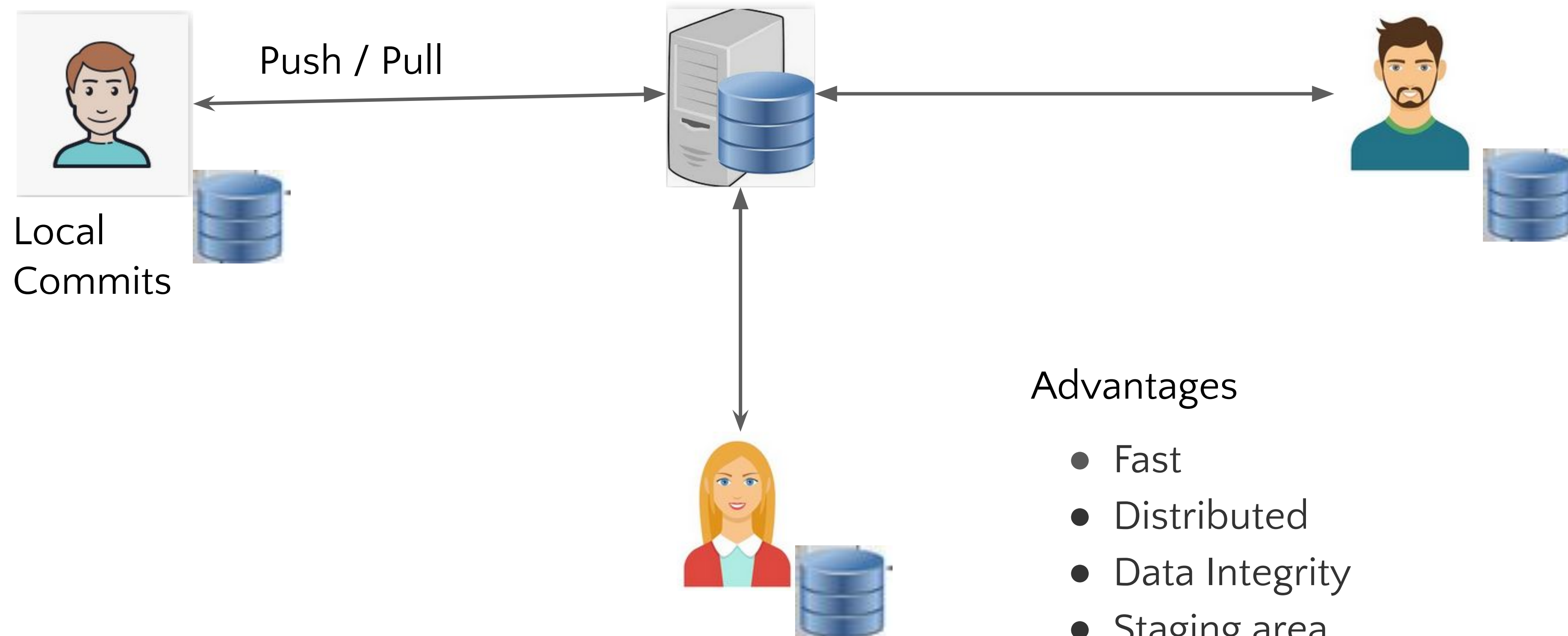
Version Control Centralized



Challenges

- Speed
- Offline Access

Version Control – Distributed



Advantages

- Fast
- Distributed
- Data Integrity
- Staging area
- Open Source and free

Distributed

Version Control

What is Version Control



Manages changes to a set of files over time. It also does

- Backup
- Maintains History about changes
- Compare Codes
- Experiment
- Collaborate

Lab 1 – Installing Git

- Task 1 – Installing locally – Download the git from below link

<https://git-scm.com/downloads>

Run the command **git version** (to confirm that installation is successful)

sample output : git version 2.20.0

- Task 2 – Sign up in GitHub / GitLab / BitBucket for GitUI

Setup up ssh access from command line . Generate ssh key and register with ssh-agent

Run the command to test ssh is setup correctly

```
git clone git@github.com:singh-ashok25/webserver.git
```

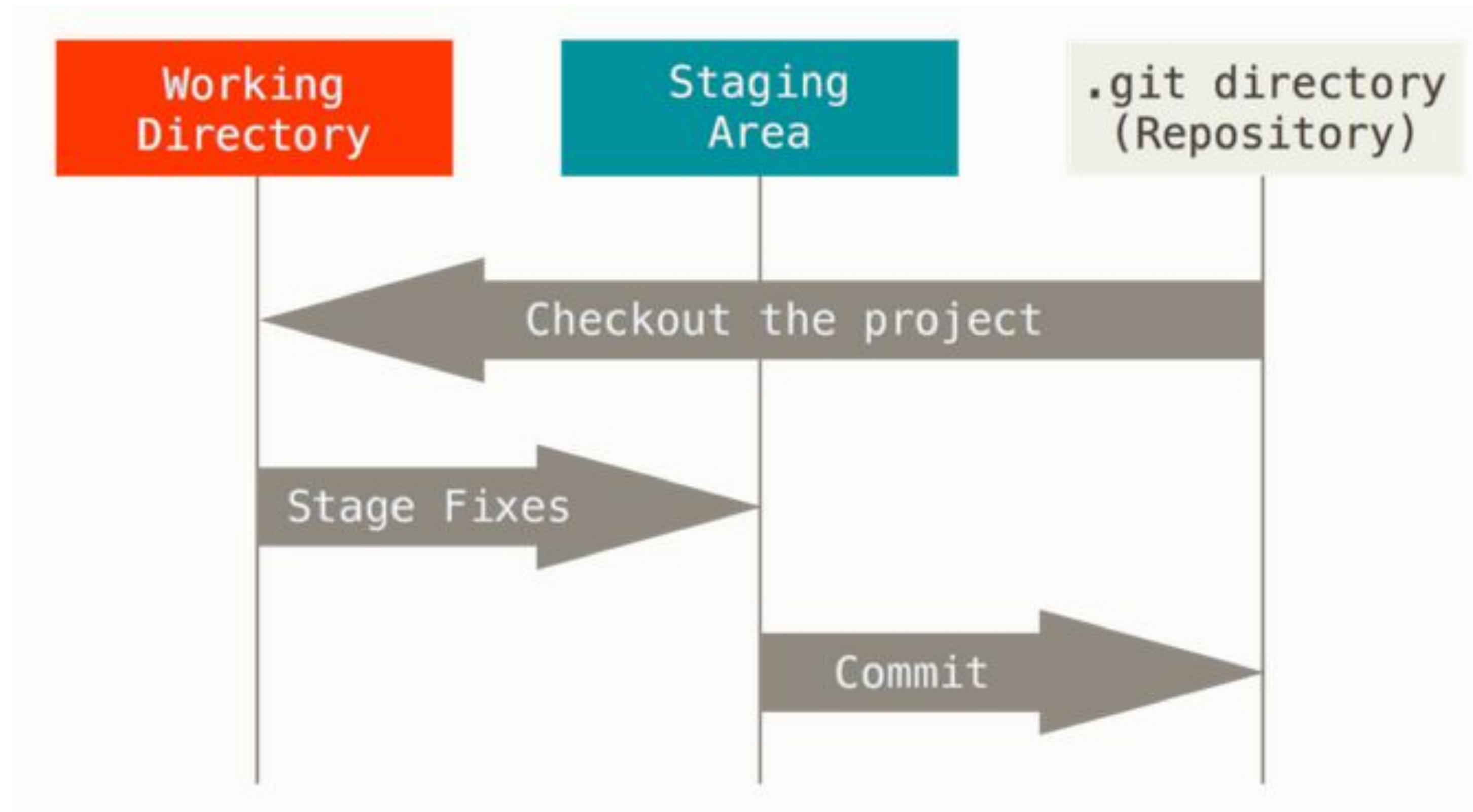
- On successful completion of lab you have
 - Git installed on host machine
 - Login access to GitHub or BitBucket (we will use GitHub in current exercise)



Generating ssh keys on windows

- Using Gitbash window
 - `ssh-keygen -t rsa -b 4096` # will generate ssh key
 - it is advisable to generate atleast 4096 bit key
- Using Puttygen
 - Launch Puttygen , Click on Generate key and move mouse point in white area.
 - Save private and public in safe location
- Copy content of file `.ssh/id_rsa.pub` to ssh key page of GitHub
- Run `git clone` command test the key

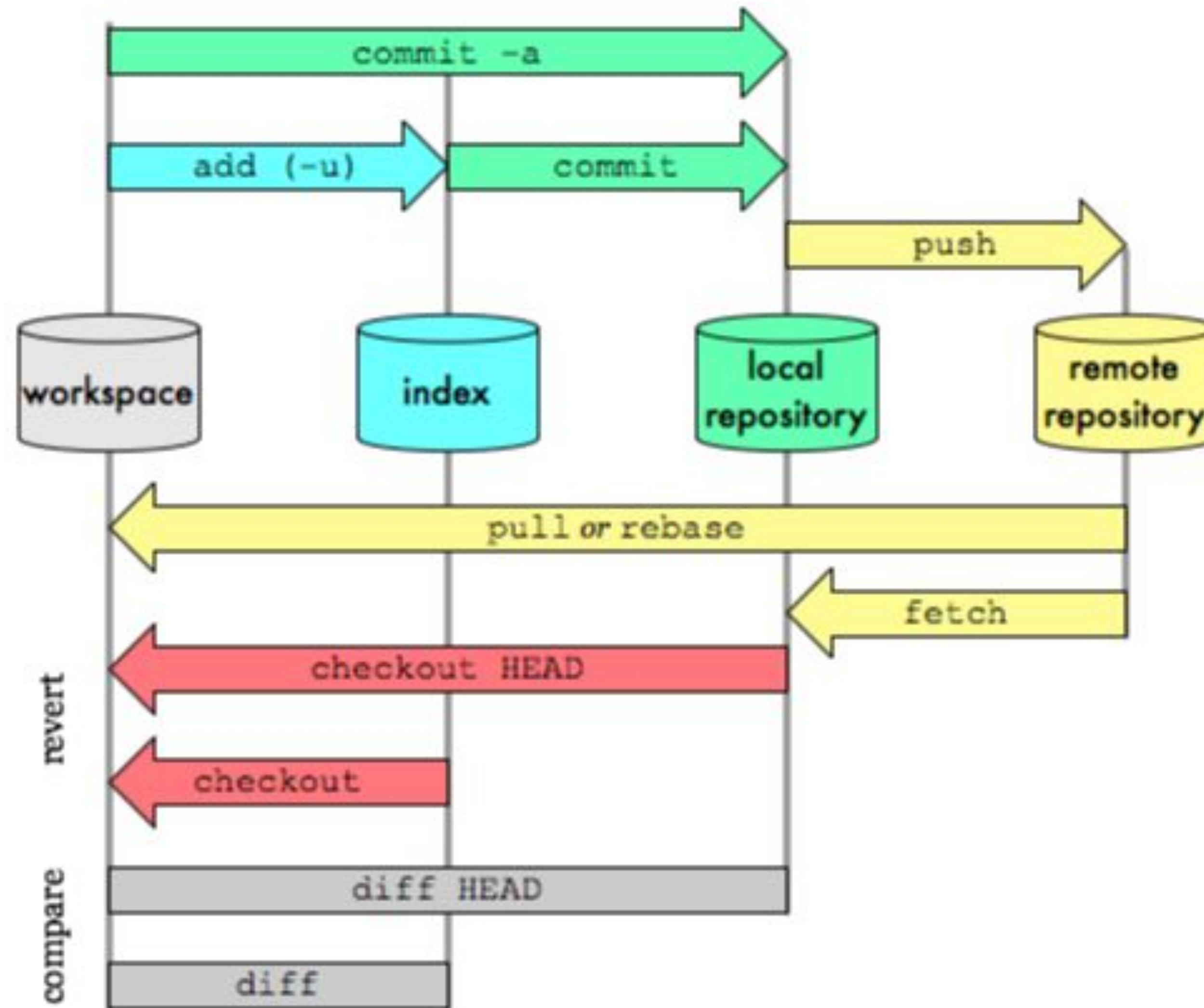
Advantages of Git



Git benefits

- Performance
- Security
- Flexibility
- Functionality

Working with Git – DataFlow



Working with Git – Setup Environment

- Git can be integrated with range of IDE like eclipse, pyCharm , VScode
- Git can also configured in shell show current repo

ashoksingh@Ashoks-Air ~/Devops-Course/GIT/LAB1/webserver (master) \$ git checkout feature/add-new-page

Switched to branch 'feature/add-new-page'

ashoksingh@Ashoks-Air ~/Devops-Course/GIT/LAB1/webserver (feature/add-new-page)

ashoksingh@Ashoks-Air ~/Devops-Course/GIT/LAB1/webserver (feature/add-new-page)* \$

- Below code will set the shell prompt environment .

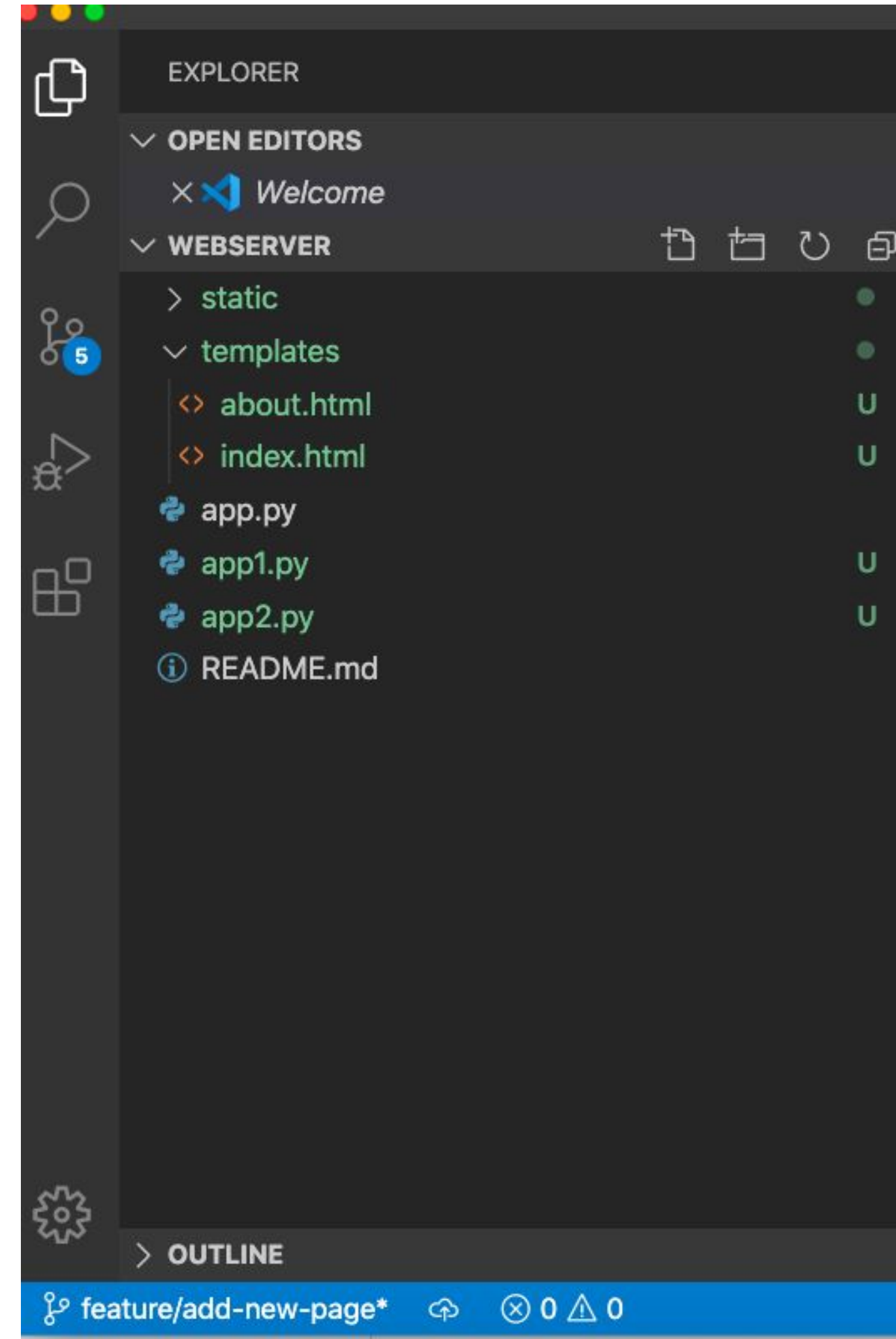
```
parse_git_branch() {  
    git branch 2> /dev/null | sed -e '/^[^*]/d' -e 's/* \(.*\)/ (\1)/'  
}  
  
parse_git_dirty() {  
    [[ -n "$(git status -s 2> /dev/null)" ]] && echo "*"  
}  
  
export PS1="\u@\h \[\033[32m\]\w\[\033[33m\]\$(parse_git_branch)\$(parse_git_dirty)\[\033[00m\] $ "
```



Working with Git – Setup VSCode Environment

VSCode is a free IDE which good features

- Install support for python language
- Shows Untracked file in Git Repo. (U)
- Working Git branch is displayed
- * Dirty branch status is shown
- Files can be added by clicking + sign which run git add command in background



Git Fundamental

Working with Git



Working with Git – Setting up Repository

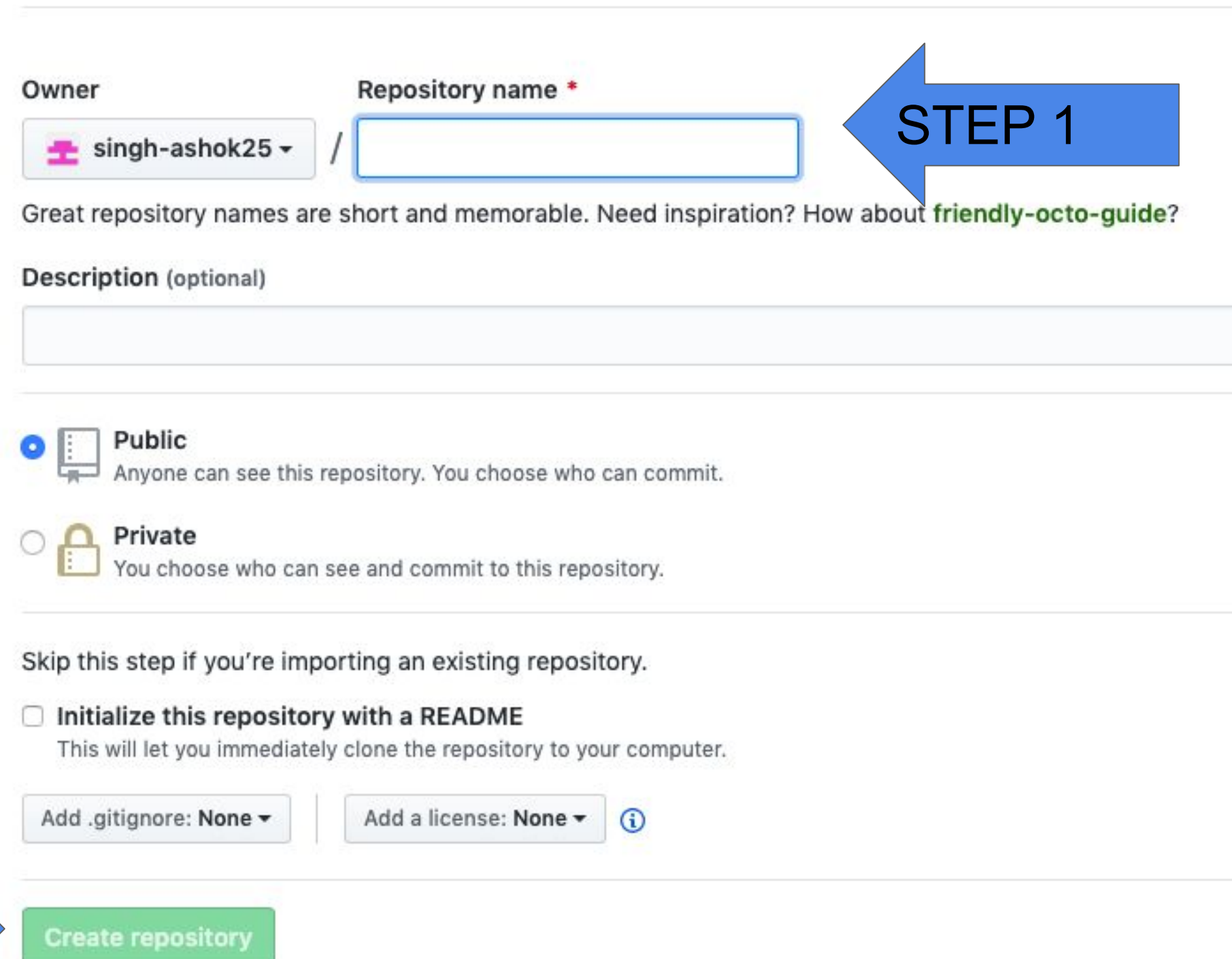
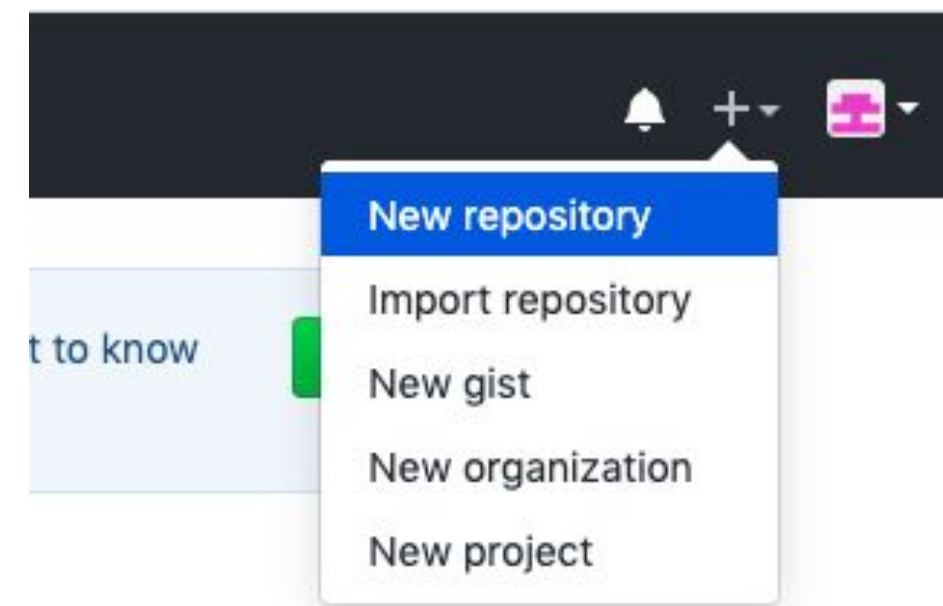
- What is a Git repository?
 - A Git repository is set of files which are needed for a project. It allows you to save versions of your code, which you can access when needed.
- `git init` can be used to convert an existing, unversioned project to a Git repository or initialize a new, empty repository. Most other Git commands are not available outside of an initialized repository, so this is the first command to be run
- The `git init` command creates a new Git repository.
 - `cd <code-directory>`
 - `git init`
 - `git remote add origin git@github.com:<username>/<repo-name>.git`
 - `git push -u origin master`



Lab 2 – Creating repository

- Objective : Creating a repository for a sample project code base in Git. Setting up development environment for IDE

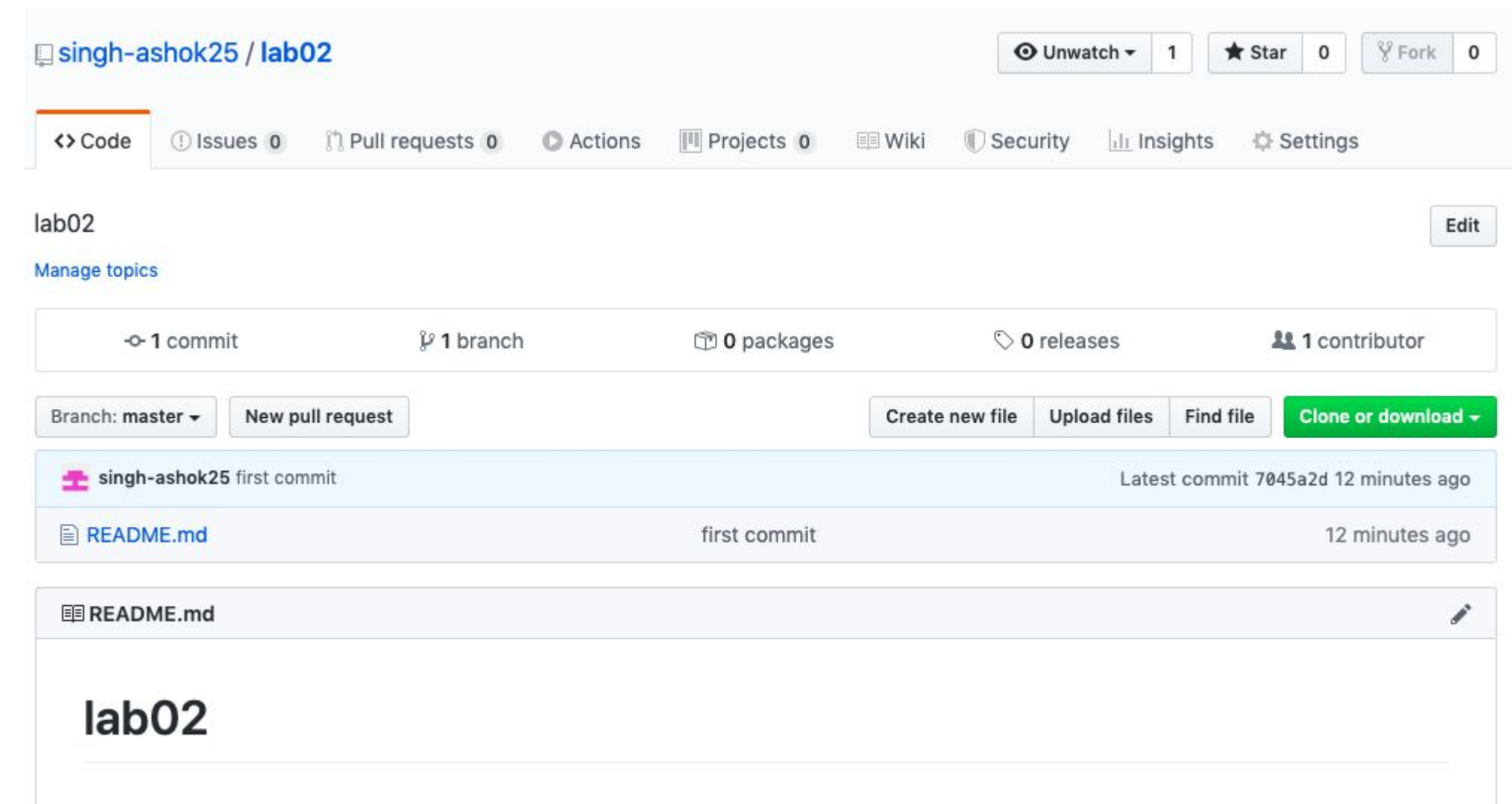
Task 1: Create New repo on Github. Do not initialize the new repository with *README*, license, or gitignore files. You can add these files after your project has been pushed to GitHub.

A screenshot of the GitHub 'Create new repository' form. The 'Owner' is 'singh-ashok25'. The 'Repository name' field is empty and highlighted with a blue border. A blue arrow labeled 'STEP 1' points to this field. Below the name field is a text prompt: 'Great repository names are short and memorable. Need inspiration? How about friendly-octo-guide?'. The 'Description (optional)' field is empty. Under 'Visibility', 'Public' is selected with a radio button. Below it, 'Private' is an option. A note says 'Skip this step if you're importing an existing repository.' Below that, 'Initialize this repository with a README' is unchecked. At the bottom, there are dropdowns for '.gitignore: None' and 'Add a license: None', and a green 'Create repository' button. A blue arrow labeled 'STEP 2' points to this button.

Lab 2 – Creating repository

TASK – 2 – Run below commands

- `echo "# lab02" >> README.md`
- `git init`
- `git add README.md`
- `git commit -m "first commit"`
- `git remote add origin git@github.com:<user_name>/<repo_name>.git`
- `git push -u origin master`
- On successful completion of lab you have
 - Git repository created for your project
 - IDE or shell prompt displaying git status



Working with Git – Setting up Repository

- Git Config levels and files
 - local
 - By default, git config will write to a local level if no configuration option is passed
 - global
 - Global level configuration is applied to an operating system user. Global configuration values are stored in a file that is located in a user's home directory.

```
git config --global user.email "your_email@example.com"
```

```
git config --global user.name "<github username>"
```
 - system
 - System-level configuration is applied across an entire machine. This covers all users on an operating system and all repos. .



Working with Git – Cloning a repository

- **Git Clone** : Git command line utility create a exact copy of repository on local host. Git clone allows quick development collaboration
- Clone git repository
 - `git clone <repo>`
- Cloning to a specific folder
 - `git clone <repo> <directory>`
- Cloning a specific tag
 - `git clone --branch <branch_name> <repo>`
- Shallow clone
 - `git clone --depth=1 <repo>`



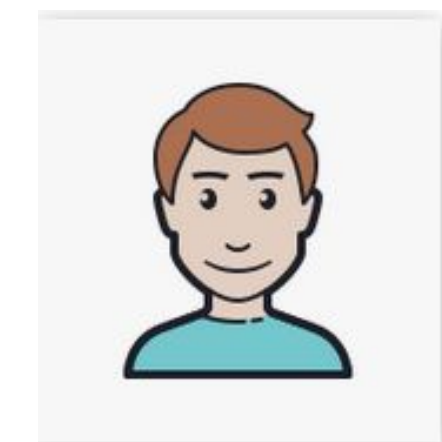
Lab 3 –Using a public repository for local development

- Objective : Use available code base in the public repository for you task customise it for individual need.
- Task 2– Login to GitHub and **clone** any public repository .
 - for the purpose of lab we will be forking using GitHub Repo
<https://github.com/singh-ashok25/webserver>
 - Install Python3 & Install Flask (<https://www.python.org/downloads/>)
 - git clone <repo>
 - python3 app.py
- On successful completion of lab you have
 - Working webserver with message “**Welcome to CloudxLab - Flask Website**”



Working with Git – Working Area

- Add new file/files in repository
 - `git add <new-filename/new-directory>`
- Updating file/files in repository
 - `git commit -m "<comment>" <filename>`
- Deleting file/files in repository
 - `git -rm -f <filename>`
- Stash changes
 - `git stash`



Local
Commits

Working with Git – Status

- Git Status : Status displays the state of the working and staging area. Status command output includes relevant instructions for staging/unstaging files. Generally there are 3 categories of status
 - Changes to be committed:
 - Changes not staged for commit:
 - Untracked files:

On branch feature/ashok-page

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: newfile

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: README.md

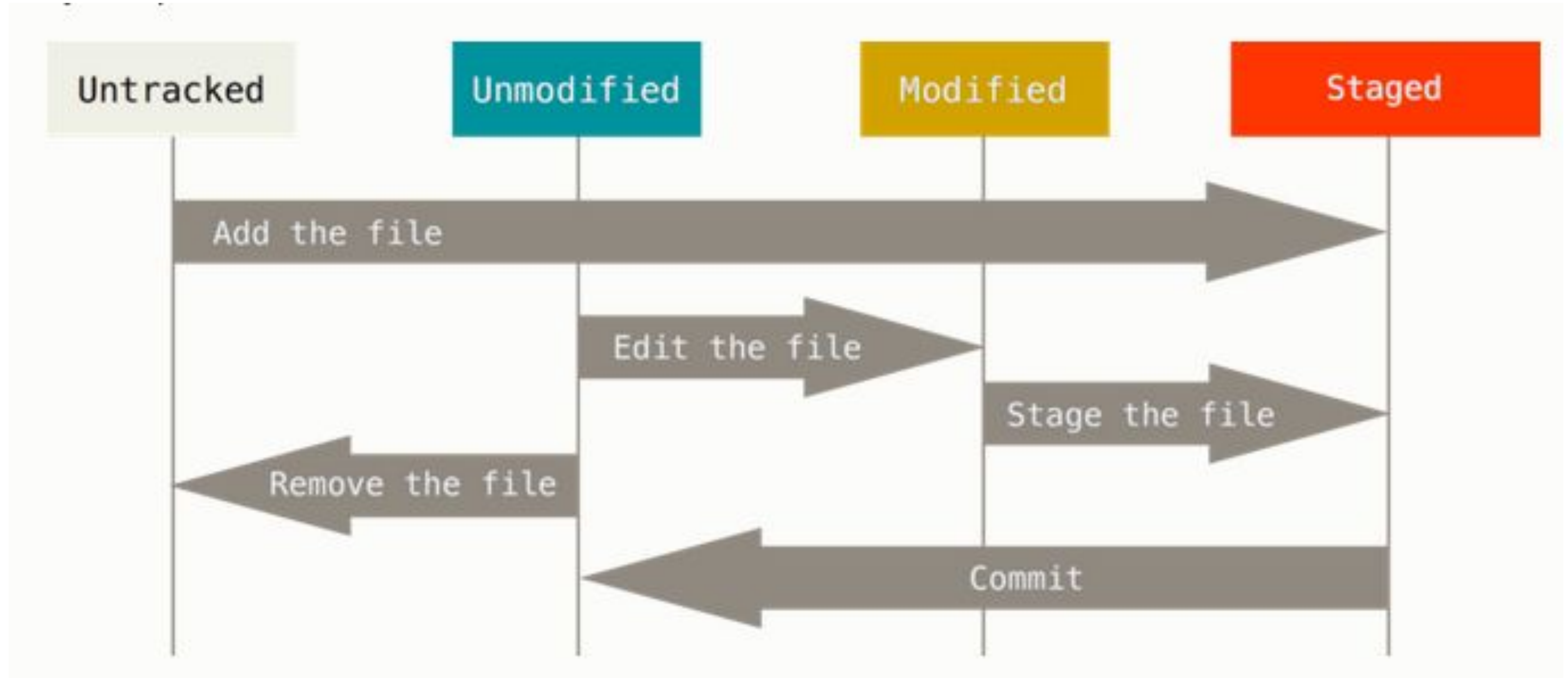
Untracked files:

(use "git add <file>..." to include in what will be committed)

newfile2



Git File Status



Working with Git – Tagging

- Git Tag: Tags are used to bookmark commits. there are 2 types of tag. Tagging are used extensively for Release management process . Tag also helps for development reporting
 - Annotated tags :Annotated tag store metadata about the tag like the commit objects
 - `git tag -a v1.4 -m "Comments"`
 - Lightweight tag :Lightweight tags create a new tag checksum and store it in the .git/ directory of the project's repo.
 - `git tag v1.4-lw`
- Listing Tags and deleting Tas
 - `git tag`
 - `git tag -d <tag name>`
- Pushing Tags :Sharing tags is similar to pushing branches. By default, git push will not push tags. Tags have to be explicitly passed to git push.
 - `git push origin v1.4`



Working with Git – Undoing Commits

- Git checkout :
 - Switch branches or restore working tree files.it also updates files in the working tree to match the version in the index or the specified tree.
 - `git checkout <branch-name>`
- Git clean
 - Cleans the working tree by recursively removing files that are not under version control, starting from the current directory..
 - `git clean -i`
- git logs
 - ch
 - `git log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit`



Working with Git – Reset and remove

- Git reset
 - Reset current HEAD to the specified state. mixed reset is the default option. The three arguments each correspond to Git's three internal state management mechanism, The Commit Tree (HEAD), The Staging Index, and The Working Directory.
 - `git reset soft|mixed|hard HEAD~`
- Git rm
 - Remove files matching pathspec from the index, or from the working tree and the index. git rm will not remove a file from just your working directory.
 - `git rm <file-name>`



Collaborating with Git – Syncing

- Git fetch : The git fetch command downloads commits, files, and refs from a remote repository into your local repo. Fetching is what you do when you want to see what everybody else has been working on.
 - `git fetch [<branch-name>]`
- Git pull : The git pull command is used to fetch and download content from a remote repository and immediately update the local repository to match that content. The git pull command is actually a combination of two other commands, git fetch followed by git merge.
 - `git pull`
- Git push : The git push command is used to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to a remote repo. It's the counterpart to git fetch, but whereas fetching imports commits to local branches, pushing exports commits to remote branches.
 - `git push`

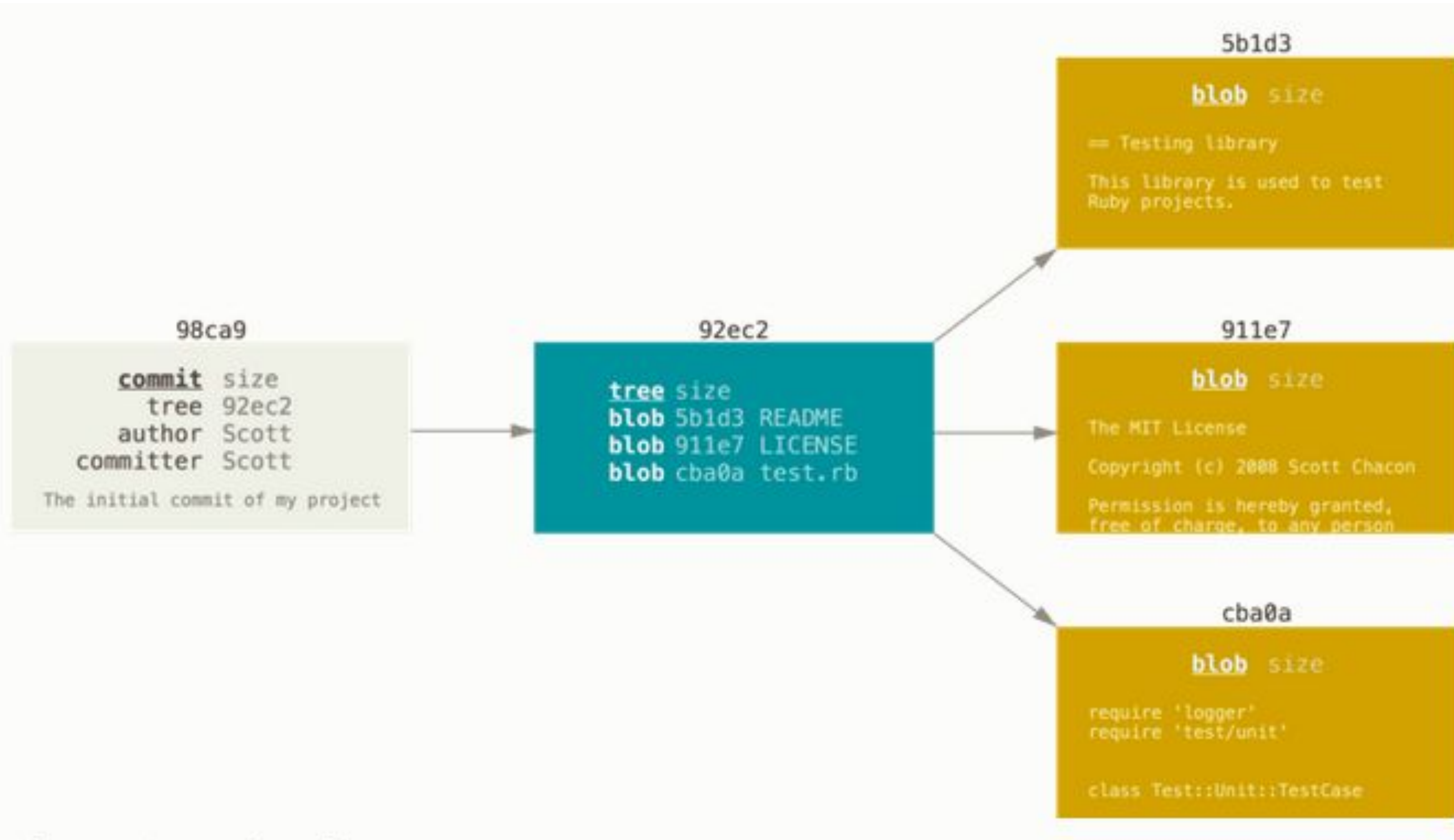


Git Branching

Collaborating with Git

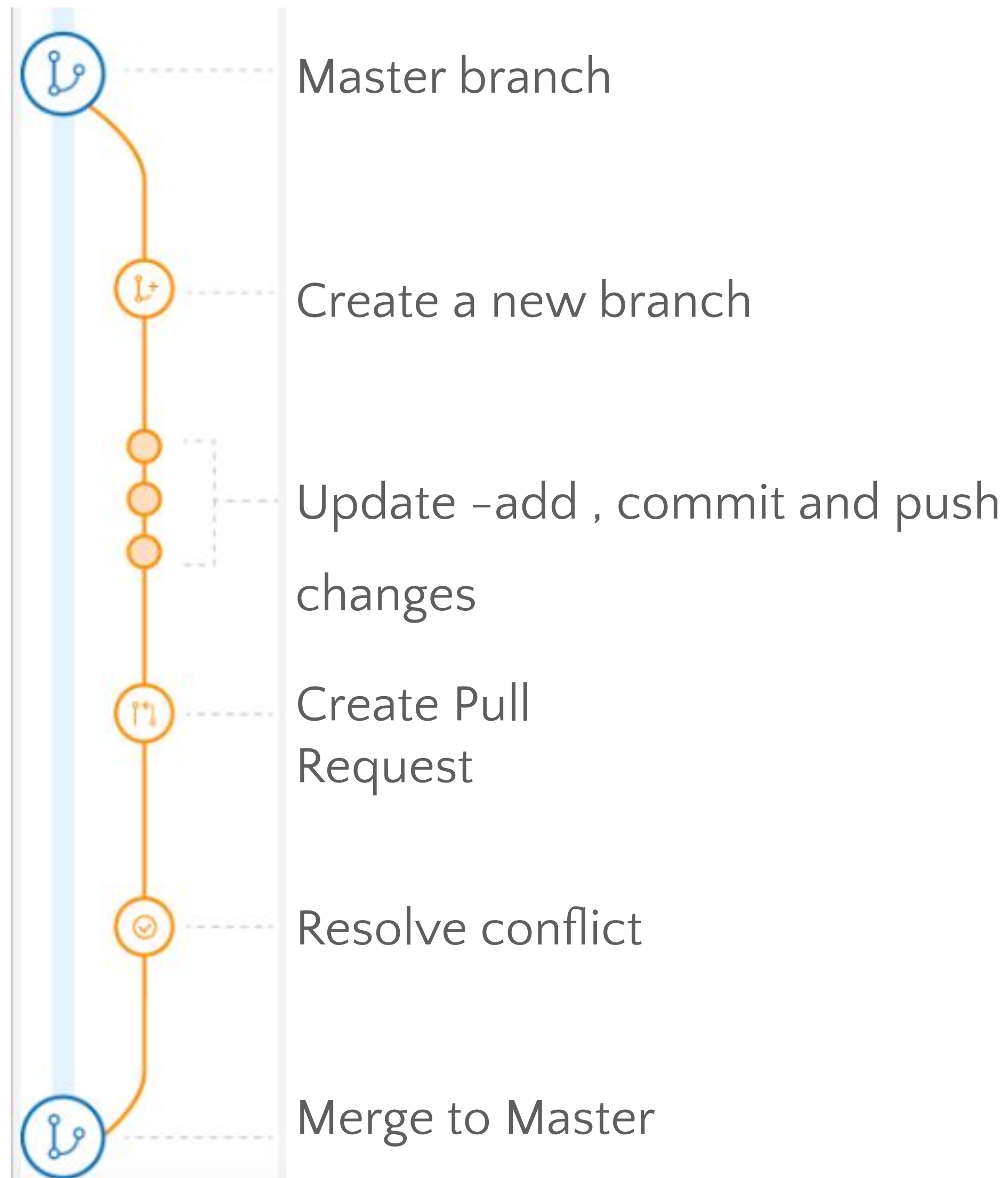


Git – Branching in a nutshell



- README
- test.rb
- LICENSE
- `git add README test.rb LICENSE`
- `git commit -m 'Initial commit'`
- 3 files and one *tree* that lists the contents of the directory and specifies which file names are stored as which blobs, and one *commit* with the pointer to that root tree and all the commit metadata

Git – Branching and Merging – Sample Workflow



- Master branch is maintains the most stable code base
- Create a new branch (for new feature , story or bug)
- Update -add , commit and push changes
 - Add new file to repos
 - Commit changes to local repo
 - push changes to server
- Create Pull Request
 - Request changes to be merged to master branch
- Review your code
- Resolve conflict
 - **Why with conflict happen ?**
- Merge to Master

Working with Git – Create Feature Branch

- Create branch
 - `git branch "<branch-name>"`
- Start working on the branch – checkout
 - `git checkout feature/<name>-page`
Switched to branch 'feature/<name>-page'
- Rebase – if master branch has changed since the feature start
 - `git rebase master origin`

Lab 4 – Developing feature to a public repository

- Objective : Developing feature in a public repository to solve a problem or add functionality
- Task
 - Create a feature branch name is “feature/<name>”
 - Switch to above branch
 - Edit index.html file in templates directory to add blue line . Relace it <your_name> with your name

```
<a href="{{ url_for('about') }}">About Us</a>
```

```
</br> <a herf="{{ url_for('<your_name>') }}"><your_name></a>
```

- cp about.html <your_name>.html (you can customised your_name.html file)
- update app2.py file as below

```
@app.route("/<your_name>.html")
```

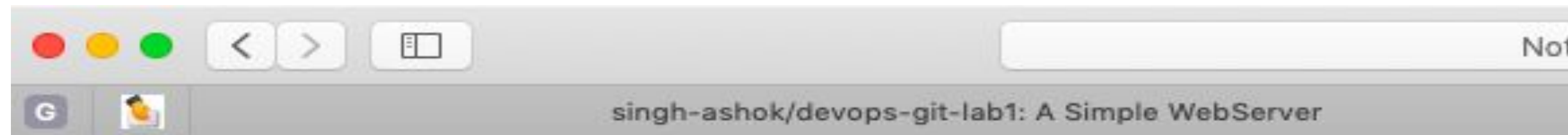
```
def <your_name>():
```

```
    return render_template('<your_name>.html')
```



Lab 4 – Developing feature to a public repository

- On successful completion of lab you will have:
 - Added a new link of your_name on main home page of web server.



Welcome to CloudxLab - Flask Website

A screenshot of the CloudxLab Flask Website homepage. The header features the CloudxLab logo, a 'Webinar on Introduction to DevOps' banner, and a 'Watch later' button. The main content area displays the webinar title 'Introduction to DevOps' and the date '22 March, 2020'. Below this, there are three speaker profiles: Ashok Singh (DevOps @Cisco), Sandeep Giri (Founder @CloudxLab), and Abhinav Singh (Co-Founder @CloudxLab). Each profile includes a photo, name, title, and a brief bio. A large play button icon is centered over the speaker profiles.

[About Us](#)
[Ashok](#)

Git Collaboration

Collaborating with Git



Collaborating with Git – Making a Pull Request

- Login to GitHub
- Click on Pull Request

The screenshot shows a GitHub Pull Request interface for the repository 'singh-ashok25 / lab2'. At the top, there are buttons for 'Unwatch', 'Star', and 'Fork'. Below this is a navigation bar with links to 'Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The main heading is 'Ashok lab2 #1'. Below the heading, it says 'singh-ashok25 wants to merge 2 commits into master from ashok-lab2'. There are tabs for 'Conversation', 'Commits', 'Checks', and 'Files changed'. The 'Conversation' tab is active, showing a comment from 'singh-ashok25' stating 'No description provided.' Below the comment, it shows two commits: 'Update README' with a 'Verified' status and commit hash 'c41fa7a', and another 'Update README' with a 'Verified' status and commit hash 'bd353c2'. On the right side, there are sections for 'Reviewers', 'Assignees', 'Labels', 'Projects', 'Milestone', and 'Linked issues'. At the bottom, there is a green box with a message: 'Continuous integration has not been set up' and 'This branch has no conflicts with the base branch'. A green button labeled 'Merge pull request' is visible.



Lab 5 – Submitting feature to a public repository

- Objective : Push local code changes to feature branch in public repository
- Task :
 - Push local repo change of new feature to server repo
 - login to GitHub and create a Pull Request
 - Once Pull Request is merged by repo owner . you changes will be publicly released.
 - validate your changes are Merged in main branch

Congratulations!! – you have contributed into a public
repo

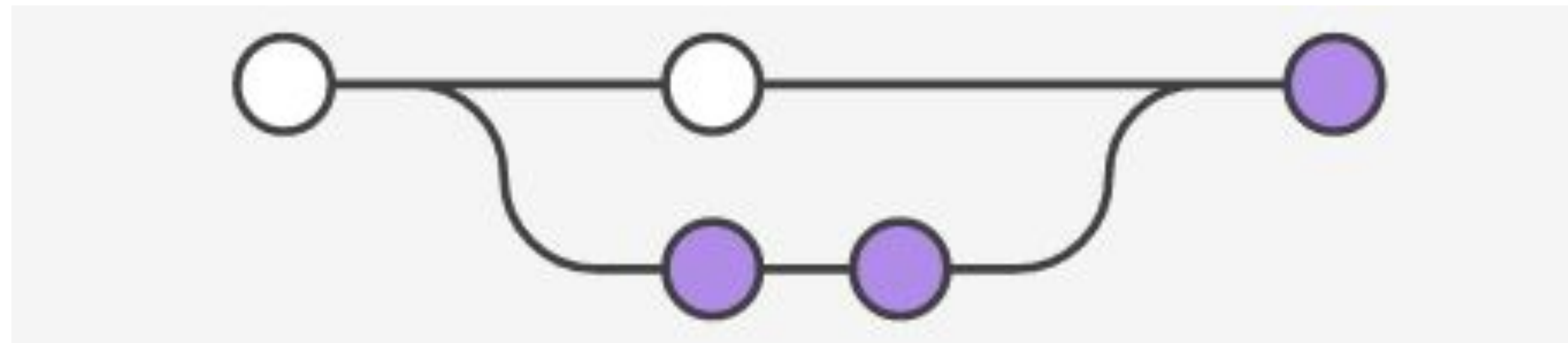
Git Workflow

Feature / Release development process

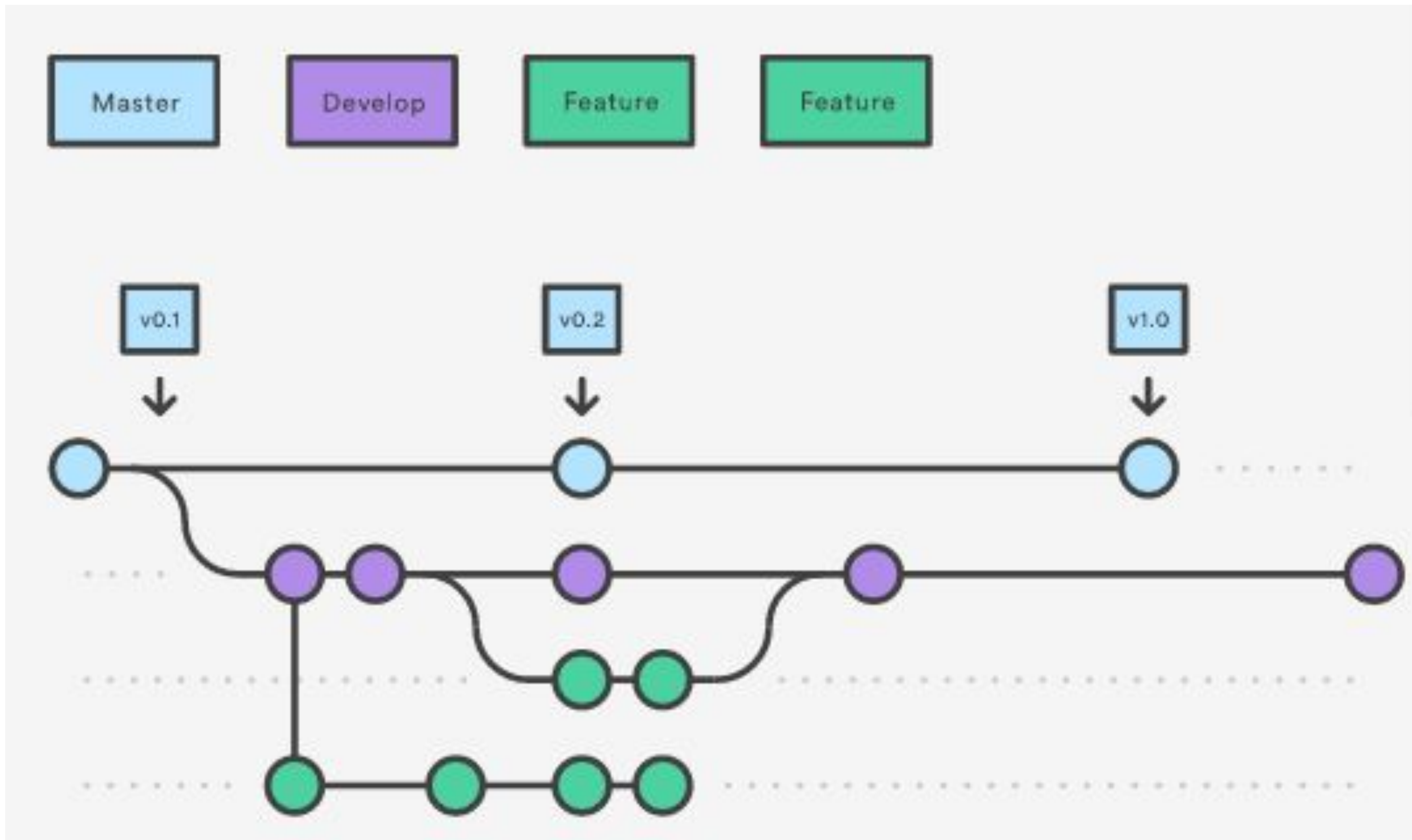


Git Feature Branch Workflow

- The core idea behind the Feature Branch Workflow is that all feature development should take place in a dedicated branch instead of the master branch.
 - 1 branch per feature
 - one developer can be working on multiple branch at same time
 - all feature merge in master after conflict resolution



GitFlow Workflow



Forking Workflow

- A developer 'forks' an 'official' server-side repository. This creates their own server-side copy.
- The new server-side copy is cloned to their local system.
- A Git remote path for the 'official' repository is added to the local clone.
- A new local feature branch is created.
- The developer makes changes on the new branch.
- New commits are created for the changes.
- The branch gets pushed to the developer's own server-side copy.
- The developer opens a pull request from the new branch to the 'official' repository.
- The pull request gets approved for merge and is merged into the original server-side repository

Git – Best practises

- Squash rebase workflow for **Git Commits**
- Feature branch should be squashed down to a single buildable commit, and then rebased from the up-to-date main branch.
 - `git pull origin master`
 - `git checkout -b branchName`
 - `git log --graph --decorate --pretty=oneline --abbrev-commit`
 - `git rebase -i`
 - `git push origin branchName --force`

Thank you



Git – squashing commits and rebase

- Create a feature branch with 4 commit and push it repo.(feature A branch)
 - `git rebase -i` (interactive)
 - squash commits
 - exit the editor and update the combined commit