git

Workshop

info@nast.edu.np

Outlines

- Getting started
- Git Basics
- Git Branching
- Github
- Git tools
- Git commands
- Git merge conflicts

Trainer

Er. Pusp Raj Joshi

Department of computer engineering Lecturer
NAST College

Sr. Project Manager at EB Pearls (3 years)
Sr. Php Developer at EB Pearls (4 years)

PHP developer IT offshore/Proshore/Stax (4 years)

https://www.linkedin.com/in/puspajoshi/

Skills:
DevOps / PHP / CSS / JS / HTML / Server Management / CMS and Frameworks / project

Management

Chapter 1: Getting started

Getting started

1. About Version Control
2. A Short History of Git
3. What is Git?
4. The Command Line
5. Installing Git
6. First-Time Git Setup
7. Getting Help
8. Summary

About Version Control

- A system that records changes to a file or set of files.
- To recall specific version later.
- If you are a graphic or web designer and want to keep every version of an image or layout, a Version Control System (VCS) is a very wise thing to use.

It allows you to:

1. Revert selected files back to a previous state,
2. Revert the entire project back to a previous state,
3. Compare changes over time,
4. See who last modified something that might be causing a problem,
5. Who introduced an issue and when, and more.
6. If you screw things up or lose files, you can easily recover.
7. In addition, you get all this for very little overhead.

Version Control Methods

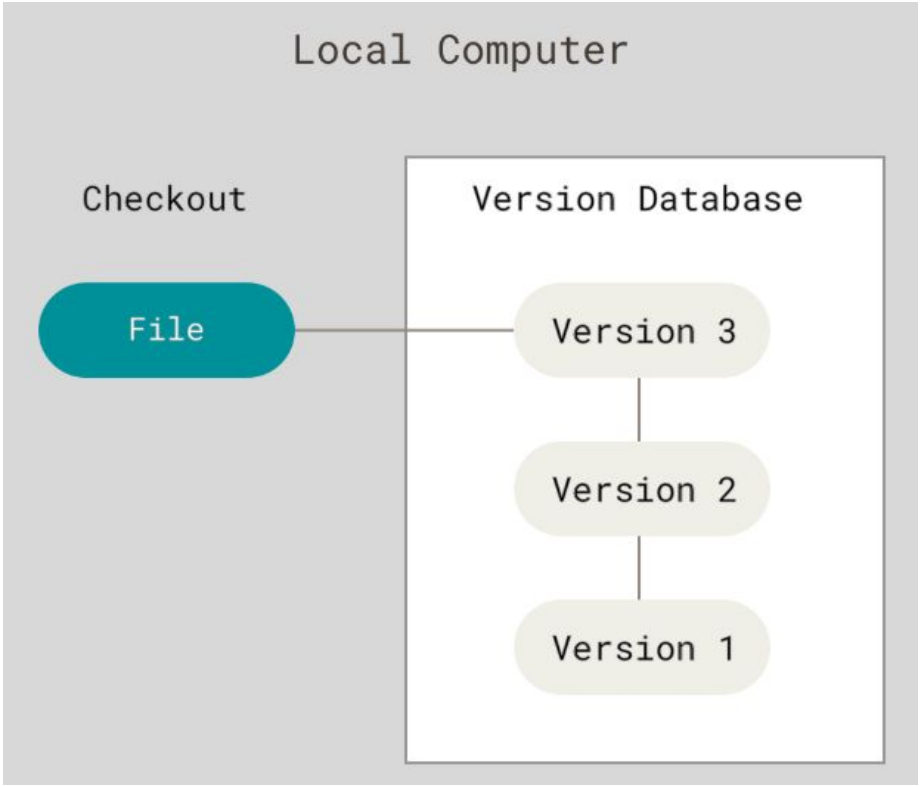
1. Local Version Control Systems
2. Centralized Version Control Systems
3. Distributed Version Control Systems

Local Version control system

Many people’s version-control method of choice is to:

- copy files into another directory.
- very common because it is so simple, incredibly error prone.
- you can forget which directory you’re in, write to the wrong file.
- To deal with this, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.
- RCS is most popular Local VCS system, which is still distributed with many computers today.
- RCS works by keeping patch sets (that is, the differences between files) in a special format on disk;
- it can then re-create what any file looked like at any point in time by adding up all the patches.

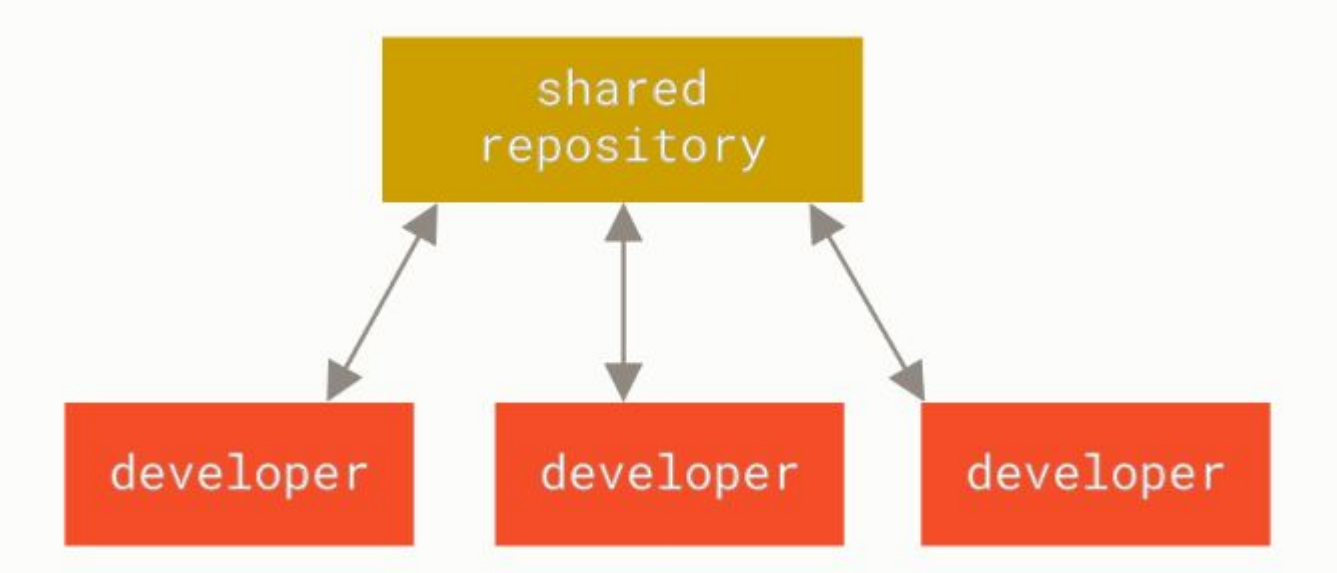
Local Version Control Systems



Centralized Version Control Systems

- Issue is that if we need to collaborate with developers on other systems. Centralized Version Control Systems (CVCSs) were developed.
- These systems (such as CVS, Subversion, and Perforce) have a single server that contains all the versioned files, and a number of clients that check out files from that central place.
- For many years, this has been the standard for version control.

Centralized Version Control Systems



Centralized Version Control Systems

Advantages over local VCSs:

- everyone knows to a certain degree what everyone else on the project is doing.
- Administrators have fine-grained control over who can do what, and it’s far easier to administer a CVCS than it is to deal with local databases on every client.

Centralized Version Control Systems

Issues of CVSs

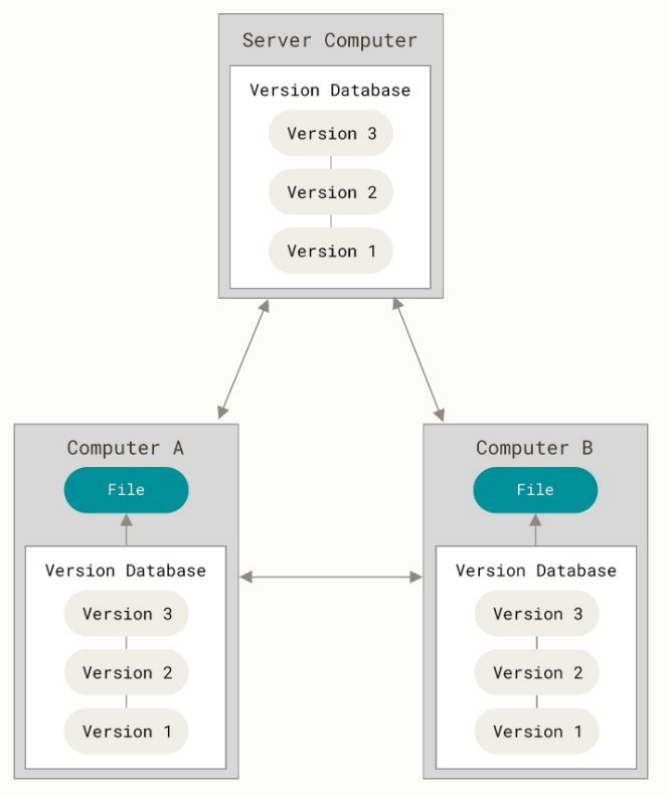
- Single point of failure that the centralized server represents.
- If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they're working on.
- If the hard disk the central database is on becomes corrupted, and proper backups haven't been kept, you lose absolutely everything—the entire history of the project except whatever single snapshots people happen to have on their local machines.
- Local VCSs suffer from this same problem—whenever you have the entire history of the project in a single place, you risk losing everything.

Distributed Version Control Systems

DVCSs

- In a DVCS (such as Git, Mercurial or Darcs), clients don’t just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history.
- Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it.
- Every clone is really a full backup of all the data.
- Furthermore, many of these systems deal pretty well with having several remote repositories they can work with, so you can collaborate with different groups of people in different ways simultaneously within the same project.
- This allows you to set up several types of workflows that aren’t possible in centralized systems, such as hierarchical models.

Distributed Version Control Systems



History of Git

- As with many great things in life, Git began with a bit of creative destruction and fiery controversy.
- The Linux kernel is an open source software project of fairly large scope. During the early years of the Linux kernel maintenance (1991–2002), changes to the software were passed around as patches and archived files. In 2002, the Linux kernel project began using a proprietary DVCS called BitKeeper.
- In 2005, the relationship between the community that developed the Linux kernel and the commercial company that developed BitKeeper broke down, and the tool's free-of-charge status was revoked. This prompted the Linux development community (and in particular Linus Torvalds, the creator of Linux) to develop their own tool based on some of the lessons they learned while using BitKeeper. Some of the goals of the new system were as follows:

History of Git (Goals of Git)

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size)
- Since its birth in 2005, Git has evolved and matured to be easy to use and yet retain these initial qualities. It’s amazingly fast, it’s very efficient with large projects, and it has an incredible branching system for non-linear development (see Git Branching).

What is Git?

- Snapshots, Not Differences
- Nearly Every Operation Is Local
- Git Has Integrity
- Git Generally Only Adds Data
- The Three States (modified, staged, and committed:)

Snapshots, Not Differences Git thinks about its data more like a stream of snapshots.

- Major difference between Git and any other VCS, Git thinks about its data.
- Other systems store information as a list of file-based changes.
- Other systems store set of files and the changes made to each file over time
- But Git thinks of its data more like a series of snapshots of a miniature filesystem.
- During every commit, or state, it takes a picture of what all your files look like at that moment.
- To be efficient, if no change in file, it doesn’t store the file again, just a link to the previous identical file it has already stored.

Snapshots, Not Differences

- important distinction between Git and nearly all other VCSs
  - most other systems copied from the previous generation.
  - Git more like a mini filesystem with some incredibly powerful tools built on top of it. (not only VCS but also a filesystem)

Nearly Every Operation Is Local

- Git need only local files and resources to operate
- CVCS where most operations have that network latency overhead.
- This aspect git is more fast because entire history of the project right there on your local disk
- To get history of project difference, it does not need server.
- You can commit at any time in your local repo, until you get network.
- This may not seem like a huge deal, but you may be surprised what a big difference it can make.

Git has integrity

- Git is checksummed before it is stored and is then referred to by that checksum.
- It’s impossible to change the contents of any file or directory without Git knowing about it.
- This functionality is built into Git at the lowest levels and is integral to its philosophy.
- You can’t lose information or get file corruption without Git being able to detect it.
- The mechanism that Git uses for this checksumming is called a SHA-1 hash.
- Git stores everything in its database not by file name but by the hash value of its contents.

Git Generally Only Adds Data

- Any changes in file, it keeps the information in git database
- It is hard to get the system to do anything that is not undoable or to make it erase data in any way.
- After commit data in git it's very difficult to lose.
- You also can recover the data that seems lost.
- Overall it adds the informations into its database.

Three states

- Modified : Modified means that you have changed the file but have not committed it to your database yet.
- Staged : Staged means that you have marked a modified file in its current version to go into your next commit snapshot.
- Committed: Committed means that the data is safely stored in your local database.

The Command Line

Why use command line ?

- There are a lot of different ways to use Git.
- We will be using Git on the command line.
- The command line is the only place you can run all Git commands—most of the GUIs implement only a partial subset of Git functionality for simplicity.
- If you know how to run the command-line version, you can probably also figure out how to run the GUI version, while the opposite is not necessarily true.

Installation

Installation

Download git first form https://git-scm.com/ and use documentation installation guide.

Every computer need git application software.

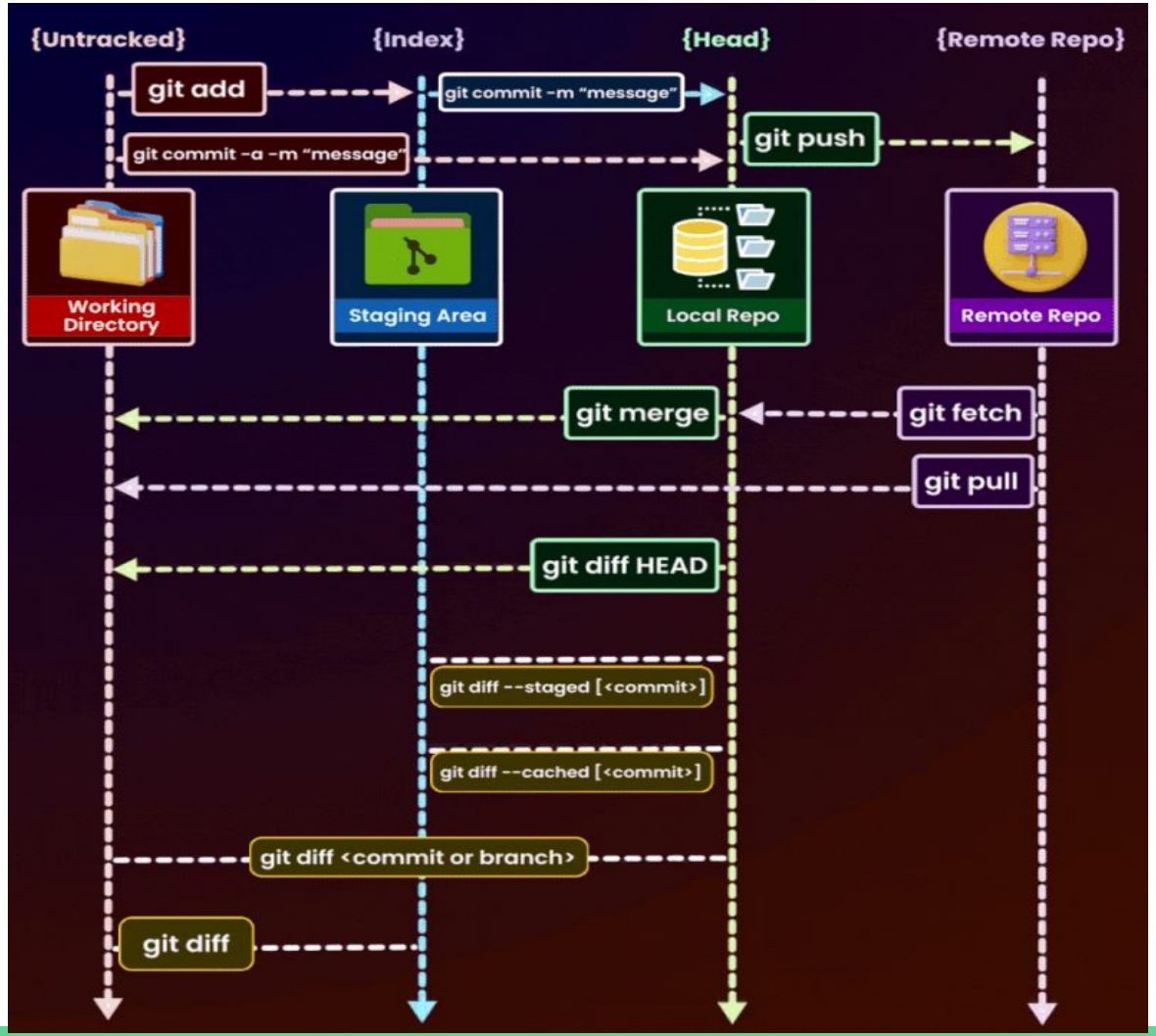
Configure Git (Information)

Git config –global user.name = ‘pusp joshi’
Git config –global user.email = ‘pusp@nast.edu.np’

git config --global init.defaultBranch main

git config --list

Git workflow



Flow

- Git add: taking a photo of a file in the way currently looks. And saved into staging area.
- Git commit: save the file in local repository
- Git push: save the file in remote repository
- Git pull: pull the file from remote repository to workstation.
- Git log: show the all log of commit

Git help

If you ever need help while using Git

- git help <verb>
- git <verb> --help
- man git-<verb>

Project Work

1. Install the git application software into your computer and configure it.
2. List all the commands that we use in future

```
start a working area (see also: git help tutorial)
   clone     Clone a repository into a new directory
   init      Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add       Add file contents to the index
   mv        Move or rename a file, a directory, or a symlink
   restore   Restore working tree files
   rm        Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect    Use binary search to find the commit that introduced a bug
   diff      Show changes between commits, commit and working tree, etc
   grep      Print lines matching a pattern
   log       Show commit logs
   show      Show various types of objects
   status    Show the working tree status

grow, mark and tweak your common history
   branch    List, create, or delete branches
   commit    Record changes to the repository
   merge     Join two or more development histories together
   rebase    Reapply commits on top of another base tip
   reset     Reset current HEAD to the specified state
   switch    Switch branches
   tag       Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch     Download objects and refs from another repository
   pull      Fetch from and integrate with another repository or a local branch
   push      Update remote refs along with associated objects
```

Importance of git in software development

Importance

1. Efficient Collaboration: multiple developers work in same codebase
2. Comprehensive Version History: maintains all the history of all changes made in codebase
3. Enhanced Code Quality: before integration peer reviews, ensuring that code is scrutinized and refined
4. Flexibility and Experimentation: Experiment in separate branch and feature in separate branch
5. Backup and Security: Every developer's local copy of the repository is a complete backup of the entire project. Git's commit integrity is secured using SHA-1 hashes, ensuring that the history of changes

github

What is github ?

1. GitHub is a code hosting platform for collaboration and version control.
2. Also GitHub is a cloud-based platform where you can store, share, and work together with others to write code.
3. GitHub lets you (and others) work together on projects.

github

Github allows us to

- Showcase or share
- Track and manage
- Code review
- Collaborate

Project Work

1. Signup github account.
2. Connect your PC with github by SSH key.

Trainer

Mukunda Bhatta

Drupal Developer at Tridz Technology (2014-2021)

Tech Lead at Elevate Services (2021 - Present)

Skills:
PHP/Drupal/Moodle/MySQL/Docker/AJAX/JS/JQuery/
Acquia Site Studio/AWS/Scrum

Git Basics

git Basics

1. Getting a Git Repository
2. Recording Changes to the Repository
3. Viewing the Commit History
4. Undoing Things
5. Working with Remotes
6. Tagging
7. Git Aliases
8. Summary

Getting a Git Repository

You typically obtain a Git repository in one of two ways:

1. You can take a local directory that is currently not under version control, and turn it into a Git repository, or
2. You can clone an existing Git repository from elsewhere.

Git initialization

You can take a local directory that is currently not under version control, and turn it into a Git repository.

```
$ git init

$ git add *.c

$ git add LICENSE

$ git commit -m 'Initial project version'
```

Cloning an Existing Repository

The command you need is git clone.

```
git clone https://github.com/libgit2/libgit2
```

That creates a directory named libgit2, initializes a .git directory inside it, pulls down all the data for that repository.

```
$ git clone https://github.com/libgit2/libgit2 mylibgit
```

That command does the same thing as the previous one, but the target directory is called mylibgit.

Project Work

1. Please group yourself so that 2 or more than 2 people can sit together.
2. Create a repository in github by only one people in github.
3. Clone that repository in your computer.

Recording changes

States

- You can checkout the working copy in your local workstation
- Two states
  - Tracked: that were in the last snapshot, as well as any newly staged files
  - Untracked : any files in your working directory that were not in your last snapshot and are not in your staging area

Viewing the commit history

git log

If you want to look back to see what has happened?

- git log lists commits
- git log -p -2 latest 2 commits with differences
- git log --stat show the status of each commit
- git log --pretty=oneline changes the format of the output log
- You can do more from git documentation

Working with remotes

Remote tools

- Github
- Bitbucket
- Gitlab
- Sourceforze
- Aws codecommit
- Azure repo
- Gitea
- Gogs

Collaboration with GitHub

About Collaboration In Personal repository

- To collaborate with users in a repository that belongs to your personal account on GitHub, you can invite the users as collaborators.
- If you want to grant more access to the repository, you can create a repository within an organization. For more information, see Access permissions on GitHub.

Inviting a collaborator

1. Ask for the username of the person you're inviting as a collaborator. username or email is required.
2. On GitHub, navigate to the main page of the repository.
3. Under your repository name, click Settings. If you cannot see the "Settings" tab, select the dropdown menu, then click Settings.
4. Screenshot of a repository header showing the tabs. The "Settings" tab is highlighted by a dark orange outline.
5. In the "Access" section of the sidebar, click Collaborators.
6. Click Add people
7. In the search field, start typing the name of person you want to invite, then click a name in the list of matches.
8. Click Add NAME to REPOSITORY.
9. The user will receive an email inviting them to the repository. Once they accept your invitation, they will have collaborator access to your repository.

Collaborating with pull requests

- Track and discuss changes in issues, then propose and review changes in pull requests.
- Completely developed in collaboration development model

Project Work

1. Please group yourself so that 2 or more than 2 people can sit together.
2. Create a repository in github by only one people in github.
3. Clone that repository in your computer.
4. Share that repository to all other members.
5. Add .gitignore file by one user and push that file in remote repository
6. Pull that changes by other users.

Tagging

Tagging

In tagging, typically, people use this functionality to mark release points:

1. Listing tags `git tag or git tag -l “v1.*”`
2. Creating tags
    a. lightweight tags
    b. annotated tags
3. Sharing tags `$ git push origin v1.5`
4. Delete tags `$ git tag -d v1.4-lw`

Annotated tags

```
$ git tag -a v1.4 -m "my
version 1.4"

$ git tag

v0.1

v1.3

v1.4
```

Lightweight tags

```
$ git tag v1.4-lw

$ git tag

v0.1

v1.3

v1.4

v1.4-lw

v1.5
```

Git Aliases

Aliases

Git experience simpler, easier, and more familiar: aliases : aliases are something you should know about.

```
$ git config --global alias.co checkout
$ git config --global alias.br branch
$ git config --global alias.ci commit
$ git config --global alias.st status
```

Git Branching

Branching

1. Branches in a Nutshell
2. Basic Branching and Merging
3. Branch Management
4. Branching Workflows
5. Remote Branches
6. Rebasing
7. Summary

Branches

Branching means you diverge from the main line of development and continue to do work without messing with that main line.

Git's branching model as its "killer feature," and it certainly sets Git apart in the VCS community.

Concepts we need to understand

- Git commits and its parents
- A branch and its commit history
- Two branches pointing into the same series of commits
- HEAD pointing to a branch

Switching Branches

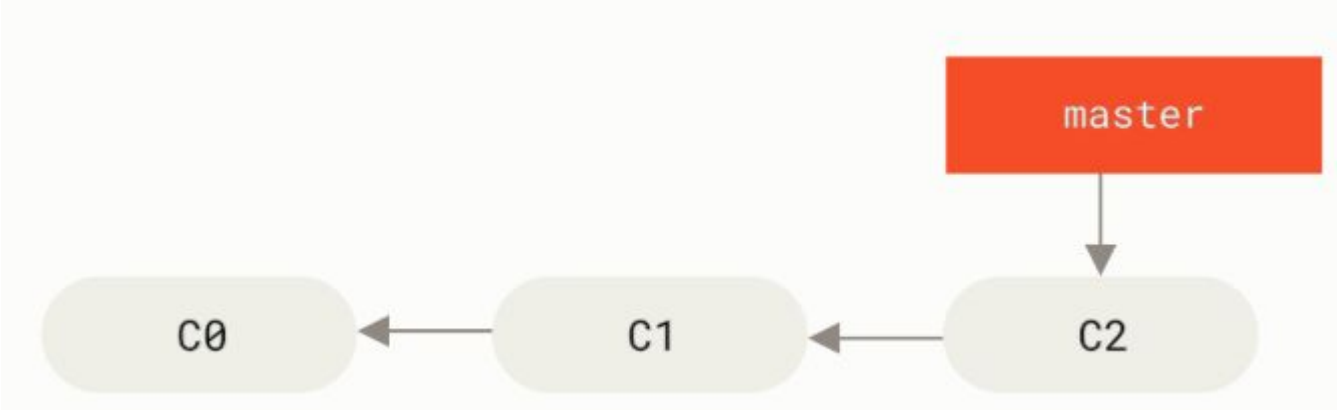
```
$ git checkout
<branch_name> or

$ git switch <branch_name>
```

See all branch graph in command line

```
$ git log --oneline
--decorate --graph --all
```

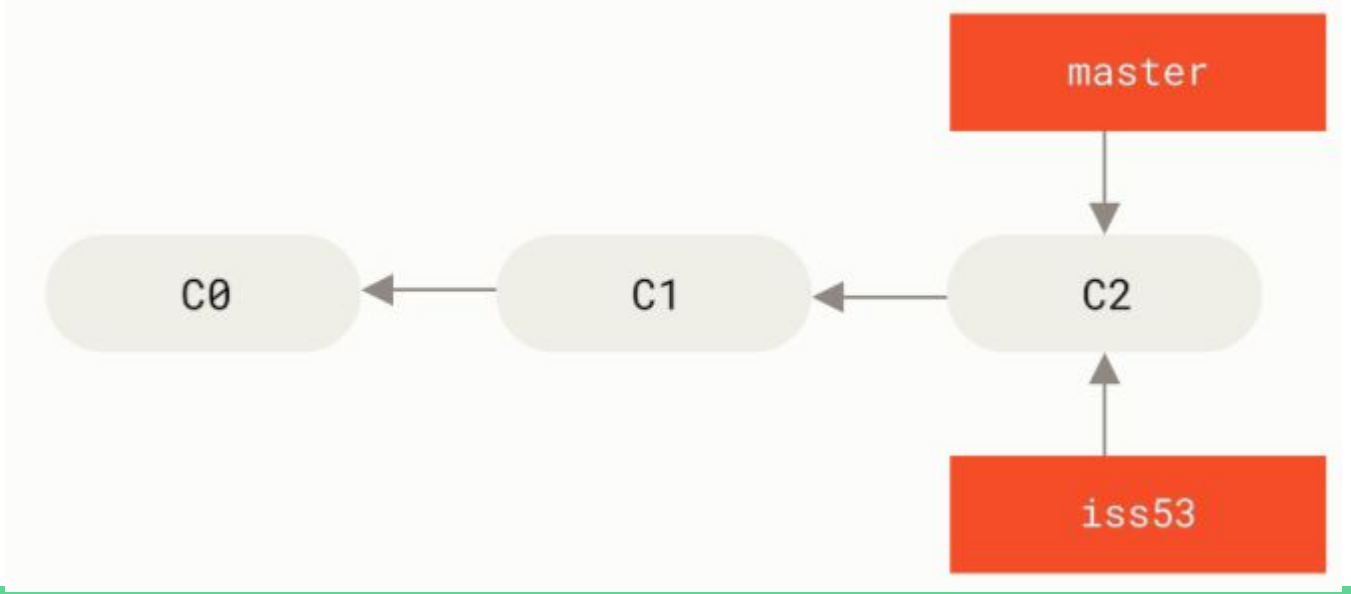
Basic Branch

1. Create branch using `$ git checkout -b <branch_name>`



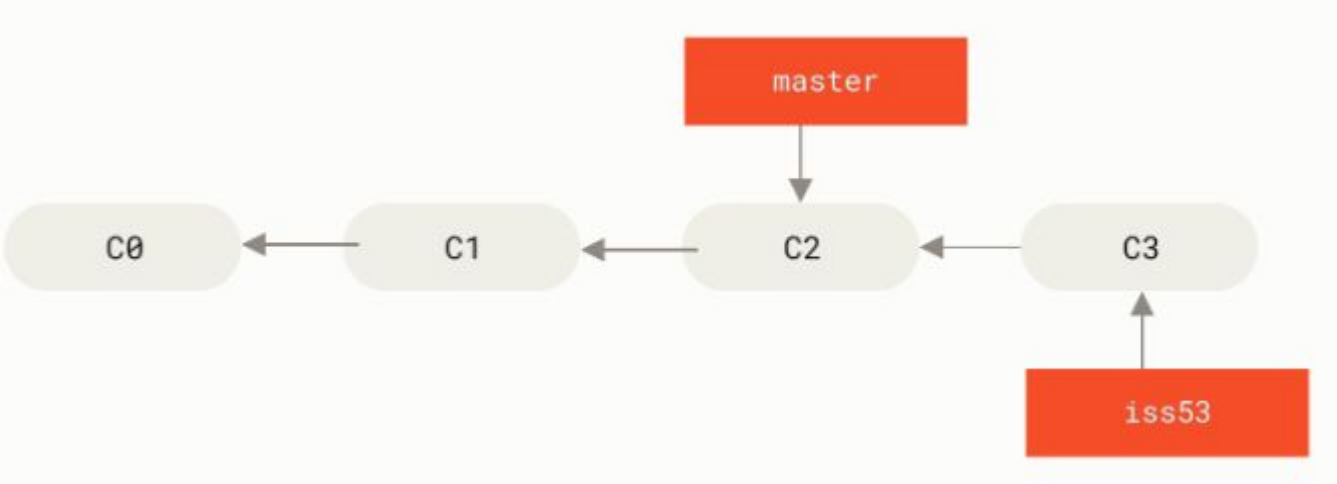
Basic Branch

1. Create branch iss53 `$ git checkout -b iss53`
2. Switched into the branch iss53 i.e Creating a new branch pointer



Basic Branch

1. Make something change in branch iss53 and commit `$ git commit -m “apply footer changes in iss53”`



Basic merge

1. Merge iss53 with master branch and delete the branch iss53

```
$ git checkout master

$ git merge iss53

$ git branch -d iss53
```

Project Work

1. You already has a repo in github
2. Now start work you all four guys in project
3. Four developers and four different tasks
    a. feature/test1
    b. feature/test2
    c. feature/test3
    d. feature/test4
4. Prepare all above branches in the local repository and push that change to own branch.

Basic Merge Conflicts

1. A conflict arises when two separate branches have made edits to the same line in a file, or when a file has been bt in one branch but edited in the other.
2. If you change the same part of the same file differently in the two branches you're merging, Git won't be able to merge them cleanly.
    a. Simultaneous Edits
    b. Conflicting Changes
    c. Complex Merges

Project Work

1. Create a more than 2 branch and change in a file in one branch second branch also change the same file in different line and in third branch delete that file. Commit the changes and merge with master branch.

Solution

1. In your team you have 3 members e.g. Ram, shyam, sita
2. Now Ram create branch ramTask in his pc from main branch
3. Shyam create branch shyamTask in his pc form main branch
4. Sita create branch sitaTask in her pc from main branch
5. They work their branch and push on their own branch and create PR.
6. During merge first branch will be successfully merged.
7. If conflict in shyamTask branch:

Solution

1. `git checkout main`
2. `git pull origin main`
3. `git checkout shyamTask`
4. `git merge main`
5. Then you will see the conflict in VS code. Resolve that conflict.
6. Then, `git add . git commit -m “conflict fix” git push origin shyamTask`
7. Then merge in github
8. Come in local `git checkout main`
9. `git pull origin main`

Branch management

Branch management

```
$ git branch : lists all branches

iss53

Branch1

* master
```

See last commit in each branch

```
$ git branch -v

iss53   93b412c Fix javascript issue

* master  7a98805 Merge branch 'iss53'

testing 782fd34 Add scott to the author list in
the readme
```

List the branch merged or not-merged

```
$ git branch –merged
```

```
$ git branch –not-merged
```

Delete the branch

```
$ git branch -d iss53
```

To remove the more set of the branches we delete the branch after they are fully merged in main or master branch

Changing a branch name

```
$ git branch --move bad-branch-name
corrected-branch-name

$ git push --set-upstream origin
corrected-branch-name

$ git push origin --delete bad-branch-name
```

example

```
$ git branch --move master main

$ git push --set-upstream origin main

$ git push origin --delete master
```

Branching Workflows practice

1. To understanding of git branching please use git visualization https://git-school.github.io/visualizing-git/

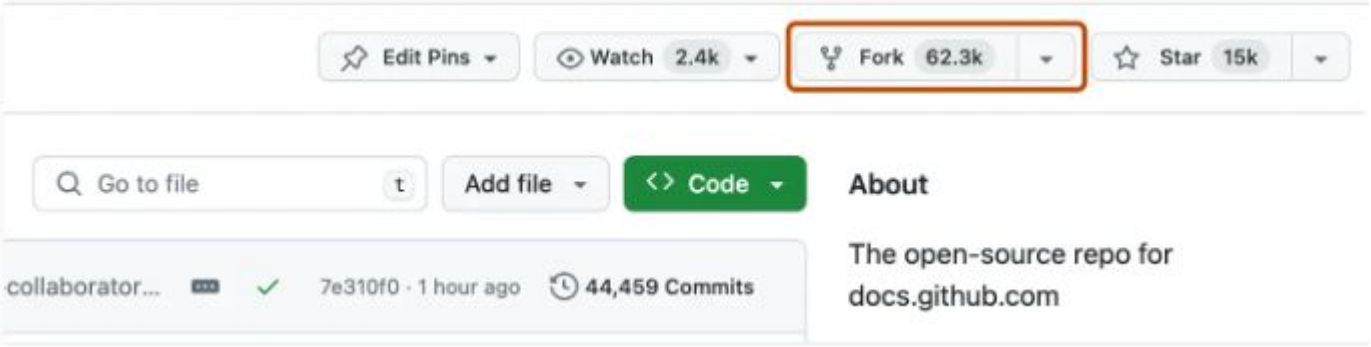
Github forking

Forking

1. A fork is a new repository that shares code and visibility settings with the original “upstream” repository.
2. Forks are often used to iterate on ideas or changes before they are proposed back to the upstream repository, such as in open source projects:
3. when a user does not have write access to the upstream repository

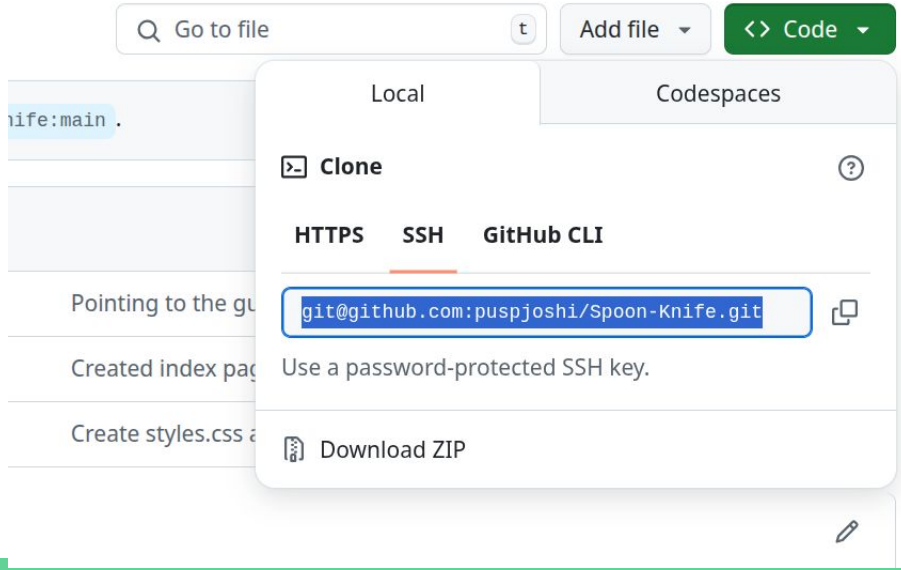
Forking a repository

1. On GitHub, navigate to the octocat/Spoon-Knife (https://github.com/octocat/Spoon-Knife) repository.
2. In the top-right corner of the page, click Fork.



Forking a repository

1. On GitHub, navigate to your fork of the Spoon-Knife repository.
2. Above the list of files, click Code. and copy the link and clone repo



Project work

1. Create 5 issues in current repo and ask the team to resolve them.

Forking a repository

1. The remote of this repository will be listed by git remote -v
2. You should have to add upstream remote
3. git remote add upstream <original repo url>
4. Now create branch
5. Add your code on that branch
6. Push the branch
7. And create PR

Work with issues

Issues

1. You can create issues in your repository to plan, discuss, and track work.
2. Issues are quick to create, flexible, and can be used in many ways.
3. Issues can track bug reports, new features and ideas, and anything else you need to write down or discuss with your team.
4. You can also break your work down further by adding sub-issues and easily browse the full hierarchy of work to be done.

Project Management with github

Project management with github

1. Using github we can track and plan our project.
2. Following are the types of project
    a. Team planning
    b. Kanban
    c. Bug tracker
    d. Iterative development
    e. Team retrospective

Project work

1. Create a issue tracker project in github and list some issue and track them each.
2. Create a team planning project in github
3. Create a kanban project in github.

GitHub Wiki

GitHub Wiki

A github wiki is a feature within GitHub repositories that allows you to create well-structured documentation alongside your code. It's commonly used for things like:

1. Project documentation
2. User guides
3. Developer notes
4. FAQs

Key Features

1. Each wiki is a separate Git repository, so it can be cloned, edited, and pushed like any repo.
2. Supports Markdown and some HTML.
3. Can be public or private (based on repo visibility).
4. Provides a simple WYSIWYG editor via the GitHub UI or raw Markdown editing.

How to Use a GitHub Wiki

1. Go to your repository on GitHub.
2. Click the **"Wiki"** tab.
3. If it's your first time, click **"Create the first page"**.

Clone the Wiki

$ git clone https://github.com/USERNAME/REPO.wiki.git

Commit and Push Updates

$ git add <File Name >

$ git commit -m "Add new documentation"

$ git pull origin <Branch Name>

$ git push origin <Branch Name>

Creating the PR (Pull request)

- Create 3 branches
- Change some files on these branches
- Push these change on your own branch
- Then create a PR and assign to main user
- Check the PR if valid merge them in master

Forking and contributing

Creating the PR (Pull request)

- Create 3 branches
- Change some files on these branches
- Push these change on your own branch
- Then create a PR and assign to main user
- Check the PR if valid merge them in master

Undoing Things

Undo

At any stage, you may want to undo something.

- amend commit
- Unstaging the staged file
- unmodifying a modified file
- Undoing things with git restore
  - Unstaging staged file
  - Unmodifying a modified file

Amend

At any stage, you may want to undo something.

```
$ git commit -m 'Initial commit'

$ git add forgotten_file

$ git commit --amend
```

Unstaging the staged file

It’s true that git reset can be a dangerous command, especially if you provide the --hard flag. However, in the scenario described above, the file in your working directory is not touched, so it’s relatively safe.

```
Git add .

Git status

Git reset HEAD <filename>
```

Unmodifying a modifying file

At any stage, you may want to undo something.

```
Git checkout – <filename>

Git checkout <filename>
```

Undoing things with git restore

- From git version 2.23.0 git introduces new command git restore

```
git restore –staged <filename>
```

Unstaging a Staged File with git restore

```
$ git add *

$ git status

$ git restore --staged
<filename>
```

Unmodifying a Modified File with git restore

We can use git restore in the place of git checkout.

```
$ git status

$ git restore --staged
<filename>
```

Project Work

1. Create folders and files
2. Add them in git
3. You only know that you added wrong file
4. Now remove that form git add (staging area)
5. Once you change above then commit the changes to local repo
6. You now understand you need to change the commit message.
7. Change the commit message and then push to github repo

References

- [1]S. Chacon and B. Straub, “Git - Book,” git-scm.com, 2014. https://git-scm.com/book/en/v2
- [2]J. Way, “Git Me Some Version Control,” Laracasts, 2025. https://laracasts.com/series/git-me-some-version-control (accessed May 18, 2025).