Version Control Systems

Git & Github

Naeem Khoshnevis

Senior Research Software Engineer
Research Computing and Data Services
Harvard University

Last Update: December 2023

Git and Github

- Distributed version control system
- Open source (https://github.com/git/git)
- Initial release: 2005

- Git host server (There are also gitlab, bitbucket)

Which approach should I use?

Graphic User Interface

Command Line Interface

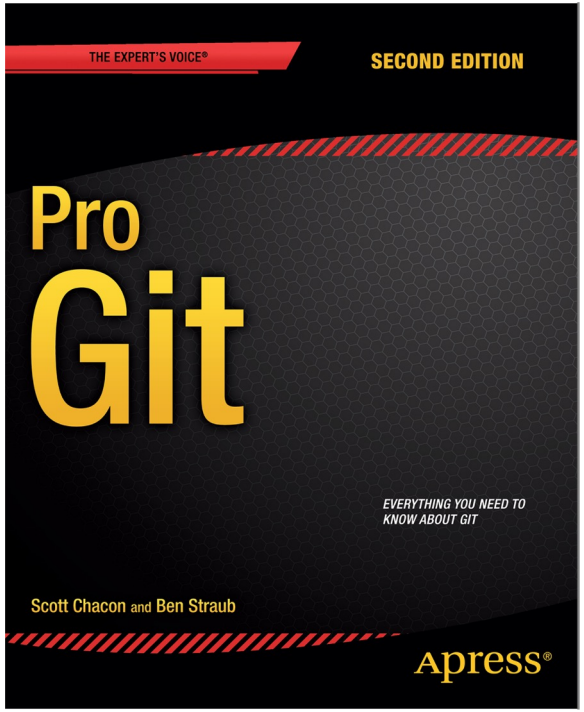
https://gitforwindows.org

Terminal

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Where to read more and get help

- Pro Git (https://git-scm.com/book/en/v2)
- git help (type `git help`)
- git help everyday (type `git help everyday`)



Different types of users

- Individual Developer (stand-alone)
  - Anybody who makes a commit, even someone who works alone.
- Individual Developer (Participant)
  - Someone who works with other people.
- Integrators
  - People who play the Integrator role.
- System administrators
  - People who are responsible for the care and feeding of git repositories.

Terminology

HEAD Special Pointer that tells git the current state of the files.

Branch Name An important value that makes the branch accessible. It points to the edge of the branch. Unlike trees, we cannot go to the branch from the root.

Master/Main The default branch, or the first branch name is master or main.

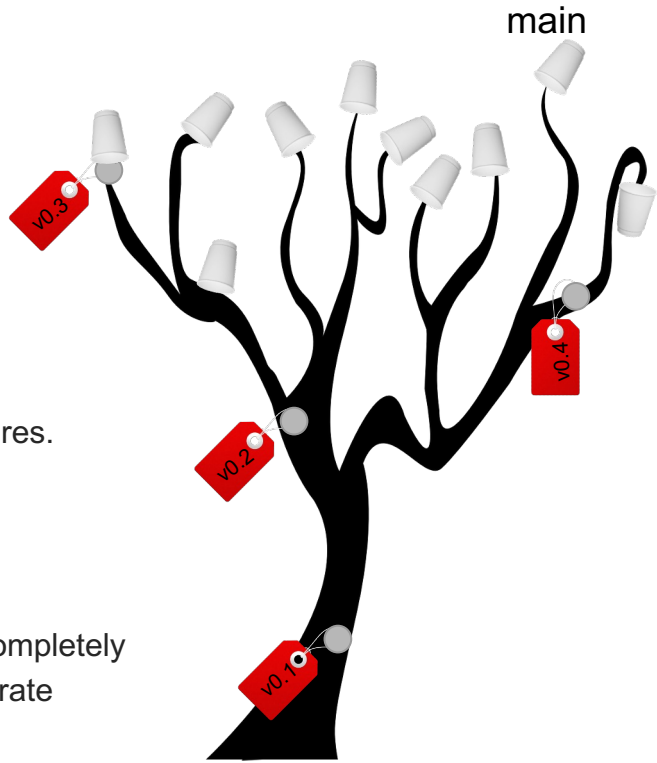
Tag Similar to branch name, however, does not move. It is permanent.

Snapshot Storing whatever is visible to the user. Similar to taking pictures.

Stage A place that you are taking snapshots.

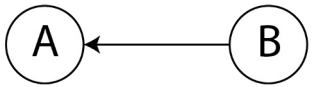
Staging Preparing files and folders to take snapshots.

remote A git repository that has a version of current document, or completely different document. It can be on the same computer or separate computers.



Terminology

- What is a commit?
- What is a commit object?
  - Git needs two identification information to accept your commits: name, and email.
- The commit object has all information to retrieve your record.
  - Who committed the record (name, and email).
  - Status of record before committing (the parent).
  - What was the committing message.



- B is built upon A.
- B is “based” on A.
- A is B’s parent.
- B = Whatever was in A + Changes that happened at the time of committing B.

Merriam-Webster SINCE 1828

commit verb

Save Word

com·mit | \ kə-ˈmit \

committed; **committing**

Definition of *commit*

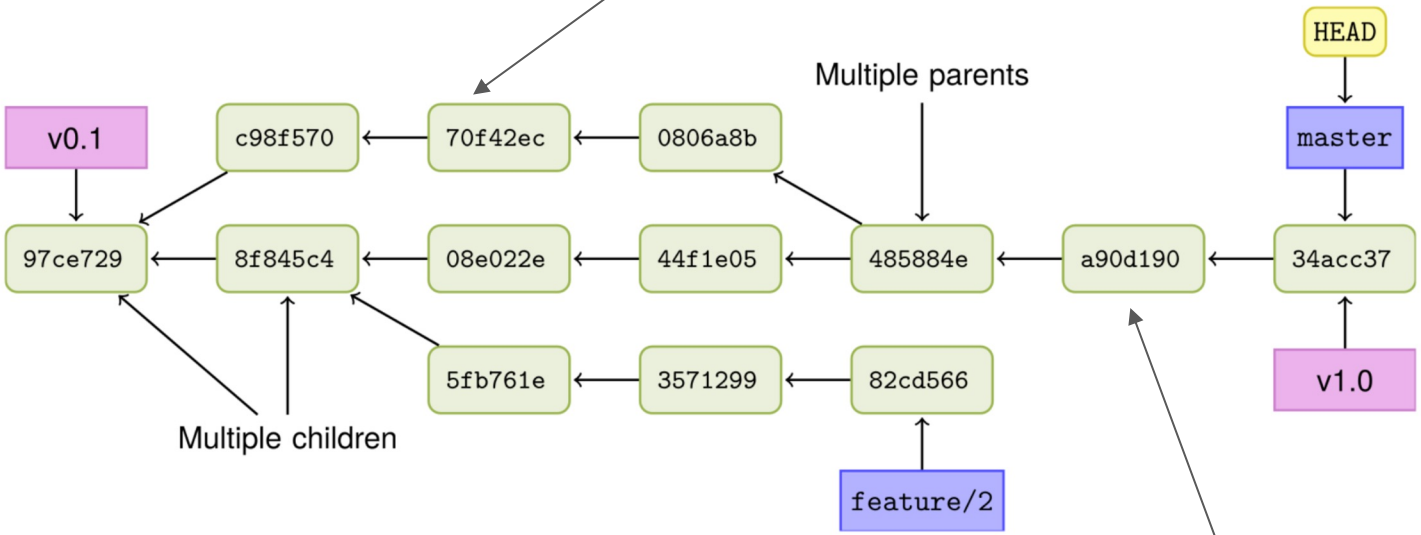
c : to consign or record for preservation
// commit it to memory

d : to put into a place for disposal or safekeeping
// The chaplain *committed* the sailor's body to the deep.

The Big Picture

Git history is a Directed Acyclic Graph (DAG)

70f42ec has all documents of c98f570 + changes that happened before committing 70f42ec (the changes are applied)

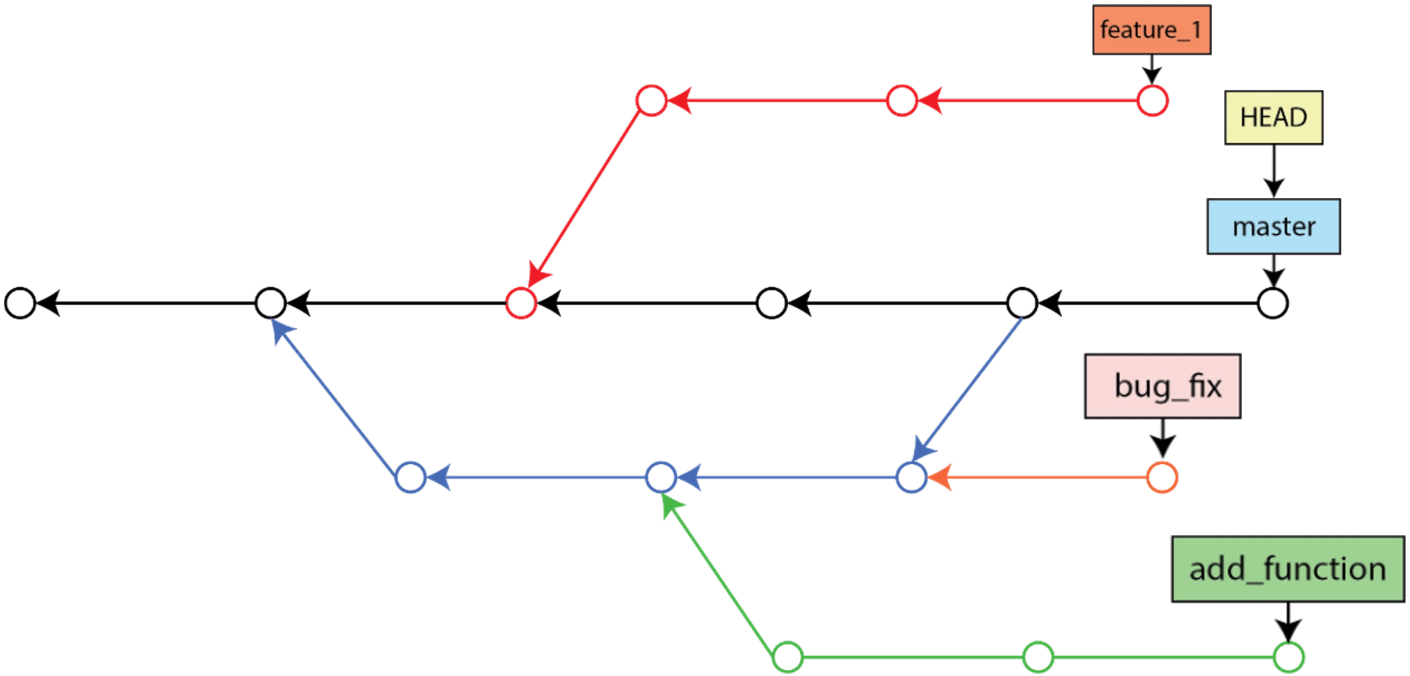


Each commit object has sufficient information to retrieve entire document that committed at the time of committing.

You can see the git history on your computer and Github.

Courtesy of https://www.oreilly.com/library/view/git-version-control/9781789137545/12ad80de-2c0d-43b6-8157-b991084640e3.xhtml

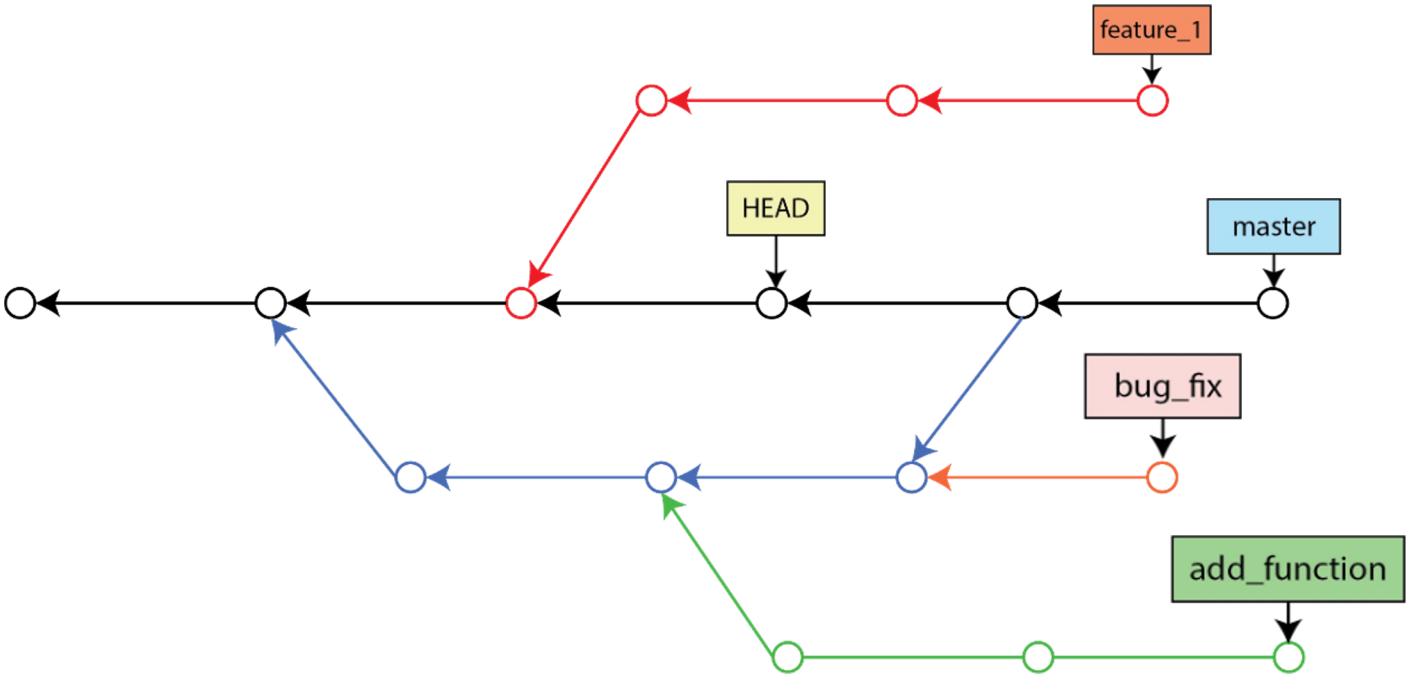
The Big Picture



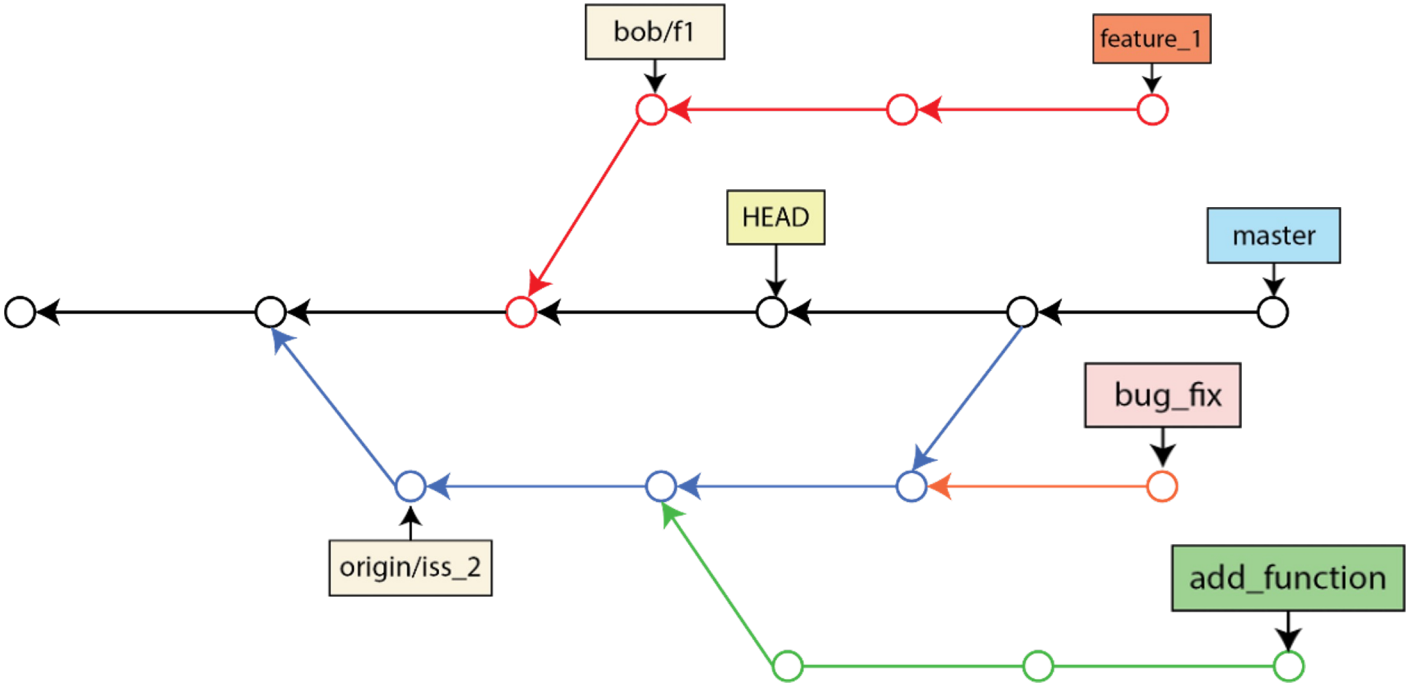
Each node stores a SHA-1 value for a commit.

Arrows always pointing to the parent or older version.

The Big Picture

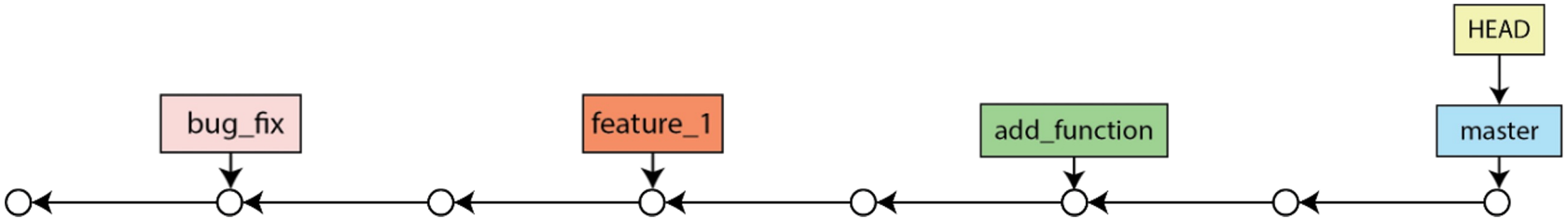


The Big Picture



A few Moments for Questions

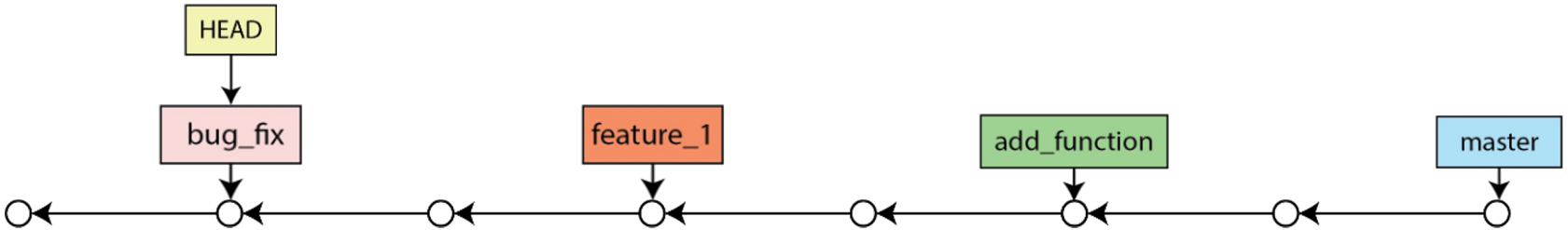
Is it possible to have the following git working history?



Scenario 1. The user started to work on the master branch; later on, he/she checked out to different commits, and started a new branches; however, after creating a new branch, he/she did not add new commits.

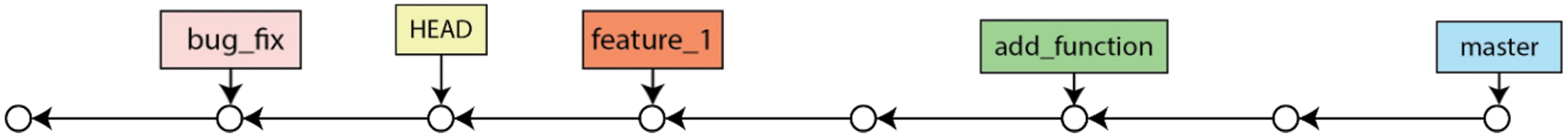
Scenario 2. The user started the bug_fix branch, then started feature_1 branch, then started the add_function branch, and then started the master branch. In this scenario, you can assume the bug_fix was the default branch.

Is it possible to have the following git working history?



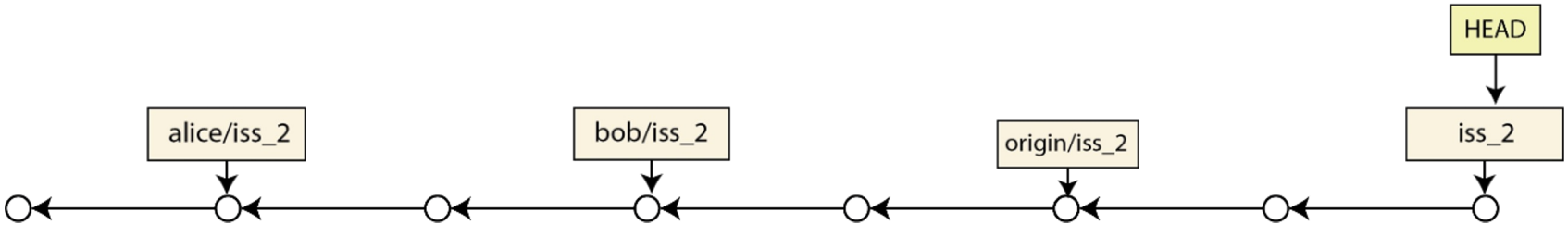
Similar to previous scenarios; however, the current state of the code is at bug_fix.

Is it possible to have the following git working history?



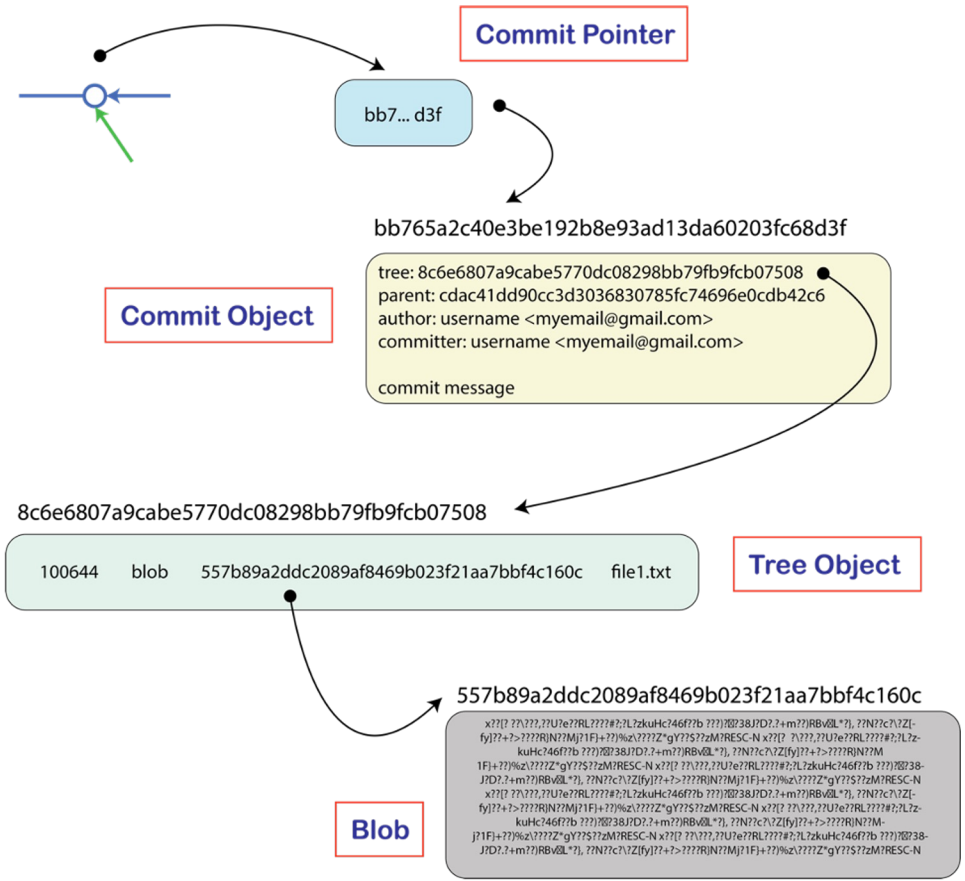
Similar to previous scenarios; however, just the user checked out to that specific commit (this is known as detached HEAD).

Is it possible to have the following git working history?



- There are three remote repositories.
- Based on convention, I assume the origin is the user's Github repo.
- Two other remote repositories are alice and bob.
- None of them have the user's current updates.

Data Structure



Commit data structure

- tree
- parent
- author
- committer

commit message

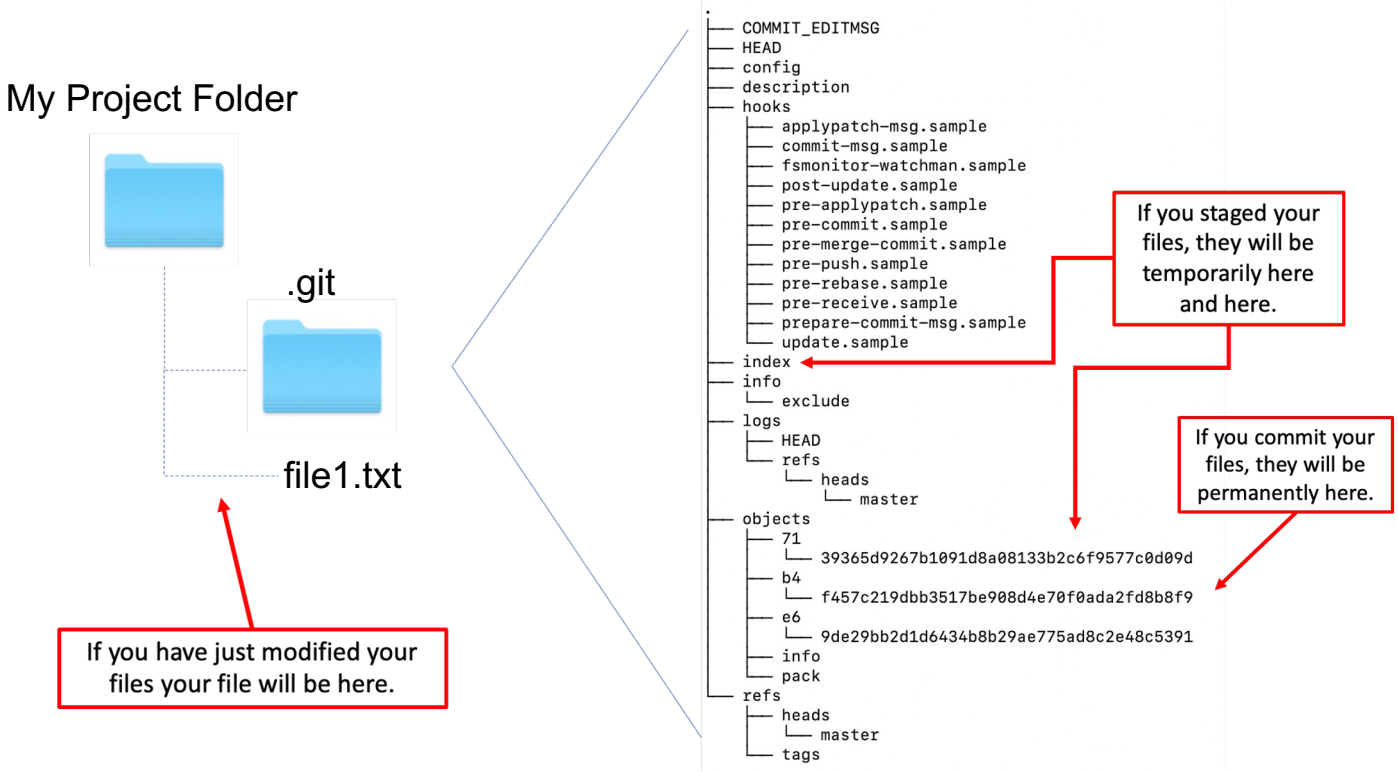
Tree data structure

- permission, type, address, filename

Blob

- binary large data object

Objects Location



A few Moments for Questions

Setup

`git config`

You can query/set/replace/unset options with this command. The name is actually the section and the key separated by a dot, and the value will be escaped.

```
$ git config --list
$ git config --global user.name "naeemkh"
$ git config --global user.email "nkhoshnevis@g.harvard.edu"
$ git config --global core.autocrlf input #(macOS and Linux)
$ git config --global core.autocrlf true #(Windows)
```

Use `--local` if you are working with different username and email address for different projects.

Setting up ssh key

If you don't already have an SSH key, you must generate a new SSH key to use for authentication. If you're unsure whether you already have an SSH key, you can check for existing keys.

https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

.gitignore file

Ignores tracking history of files based on file exact name or pattern.

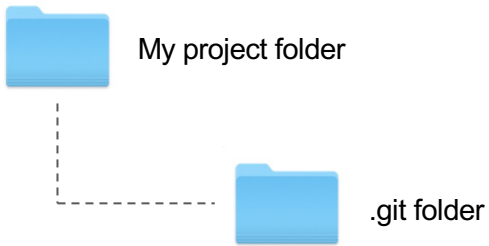
https://git-scm.com/docs/git-config

Common/everyday commands

```
git verb options
```

git init

This command creates an empty Git repository - basically a **.git** directory with subdirectories for **objects**, **refs/heads**, **refs/tags**, and template files. An initial branch without any commits will be created.



- You can convert a git repository into a ordinary directory by removing the **.git** folder.

```
$ rm -rf .git
```

git status

Shows the working tree status.

```
$ git status
$ git status --ignored
```

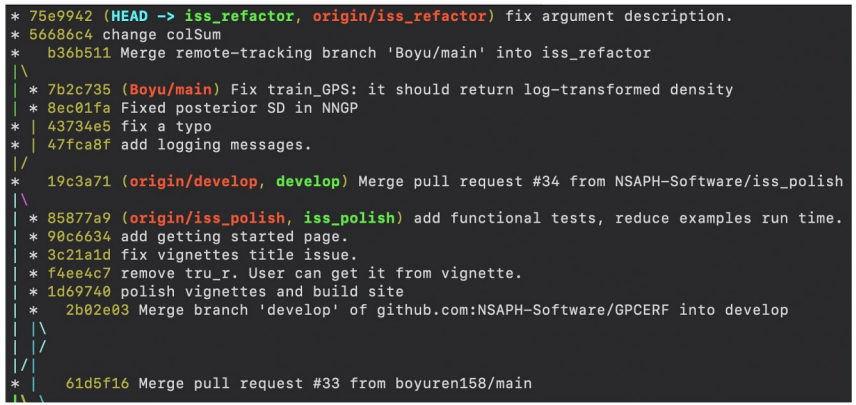
https://git-scm.com/docs/git-init
https://git-scm.com/docs/git-status

Common/everyday commands

`git log`

Shows the commit logs.

`$ git log --oneline`	Show commit as compact as possible.
`$ git log --graph`	Draw a text-based graphical representation of the commit history
`$ git log --all`	Show all commits.
`$ git log --patch file_name.txt`	Shows history of development of the file.



Common/everyday commands

`git checkout`

Updates files in the working tree to match the version in the index or the specified tree.

`$ git checkout main`	Switch to the main branch.
`$ git checkout -b new_feature`	Create new_feature branch and switch to it.
`$ git checkout c98f570`	Move HEAD to c98f570 commit.
`$ git checkout -f c98f570`	When switching branches, proceed even if the index or the working tree differs from **HEAD**. This is used to throw away local changes.
`$ git checkout v0.0.1`	Switch to a commit based on tag name.
`$ git checkout --  myfile.txt`	Discard changes in myfile.txt in the working directory.
`$ git checkout HEAD myfile.txt`	Discard changes in myfile.txt in the working directory.

Common/everyday commands

`git add`

This command updates the index using the current content found in the working tree, to prepare the content staged for the next commit. It typically adds the current content of existing paths as a whole, but with some options it can also be used to add content with only part of the changes made to the working tree files applied.

`$ git add file1.txt file2.txt`	Stage file1.txt and file2.txt for the next commit.
`$ git add -A`	Stage all files for the next commit (not recommended)
`$ git add -u`	Stage all tracked files for the next commit.
`$ git add -f ignored_file.txt`	Staged ignored_file.txt, although I have ignored this file in my initial setup.

Common/everyday commands

`git commit`

Create a new commit containing the current contents of the index and the given log message describing the changes. The new commit is a direct child of HEAD, usually the tip of the current branch, and the branch is updated to point to it (unless no branch is associated with the working tree, in which case HEAD is “detached”).

`$ git commit -m “commit message”`	Commit the staged files.
`$ git commit --amend`	Modify the last commit’s message.

The commit message should be short, descriptive, and imperative.

Note:

You can give credit to other collaborators by using their username and email as co-author in the commit message.

```
$ git commit -m "Refactor usability tests.
>
>
Co-authored-by: name <name@example.com>
Co-authored-by: another-name <another-name@example.com>"
```

https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/creating-a-commit-with-multiple-authors

Common/everyday commands

`git branch`

If --list is given, or if there are no non-option arguments, existing branches are listed; the current branch will be highlighted in green and marked with an asterisk.

`$ git branch`	Show all local branches
`$ git branch --list`	Show all local branches
`$ git branch -r`	Show all remote branches
`$ git branch -a`	Show all remote and local branches
`$ git branch -m new_name`	Change branch name
`$ git branch -d branch_name`	Remove branch_name branch

Common/everyday commands

`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 myfile.txt
```
Remove myfile.txt and stage it to commit.

```
$ git rm --cached
```
Unstage staged files.

`git show`

Shows one or more objects (blobs, trees, tags and commits).

`git cat-file`

In its first form, the command provides the content or the type of an object in the repository.

```
$ git cat-file -t 583b735
```
Show type of the object

```
$ git cat-file -p 583b735
```
Show content of the object

Common/everyday commands

`git diff`

Show changes between the working tree and the index or a tree, changes between the index and a tree, changes between two trees, changes resulting from a merge, changes between two blob objects, or changes between two files on disk.

`$ git diff`	Show differences between local changes and HEAD
`$ git diff HEAD file_name.txt`	Show differences between local changes and HEAD in file_name.txt
`$ git diff HEAD~1 file_name.txt`	Show differences between local changes and one commit before HEAD in file_name.txt
`$ git diff 645dabd file_name.txt`	Show differences between local changes in file_name.txt and its state at 645dabd commit.
`$ git diff 645dabd d3e6216 file1.txt`	Show differences in file1.txt when it was in 645dabd and d3e6216 commits.

Common/everyday commands

`git revert`

Given one or more existing commits, revert the changes that the related patches introduce, and record some new commits that record them.

```
$ git revert b6bd7a8
```

Undo changes in b6bbd7a8 and commit.

`git restore`

Restore specified paths in the working tree with some contents from a restore source. If a path is tracked but does not exist in the restore source, it will be removed to match the source.

```
$ git restore file1.txt
```

Returns local changes in file1.txt to the HEAD state.

```
$ git restore --staged file1.txt
```

Unstage file1.txt.

`git reset`

Reset current HEAD to the specified state.

```
$ git reset --soft HEAD~n
```

Uncommit last n commits but keep the local staged changes.

```
$ git reset --hard HEAD~n
```

Uncommit last n commits and clean the working tree.

Common/everyday commands

`git merge`

Incorporates changes from the named commits (since the time their histories diverged from the current branch) into the current branch.

```
$ git merge feature_1
```

Merge feature_1 branch into the current branch.

`git tag`

Create, list, delete or verify a tag object.

```
$ git tag
```

See the list of available tags.

```
$ git tag ver.0.2
```

Add tag ver.0.2 to the HEAD.

```
$ git tag -d ver.0.2
```

Remove tag ver.0.2.

Remotes

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. Collaborating with others involves managing these remote repositories and pushing and pulling data to and from them when you need to share work.

git remote

`$ git remote`	Returns list of remotes.
`$ git remote -v`	Provides the list of remotes with more details.
`$ git remote add <name> <url/path>`	Add a remote named <name> for the repository at <url>.
`$ git remote remove <name>`	Remove a remote named <name> for the repository.
`$ git remote set-url <name> <newurl>`	Change the remote's URL/path.
`$ git remote rename <oldname> <newname>`	Change the remote name.

git clone

Clones a repository into a newly created directory, creates remote-tracking branches for each branch in the cloned repository.

```
$ git clone url
```

Be careful about the project's license. https://choosealicense.com/

Remotes

`git fetch`

Fetch branches and/or tags (collectively, "refs") from one or more other repositories, along with the objects necessary to complete their histories.

`$ git fetch`	By default the **origin** remote will be used, unless there's an upstream branch configured for the current branch.
`$ git fetch -all`	All branches and remotes will be fetched.

`git push`

Updates remote refs using local refs, while sending objects necessary to complete the given refs.

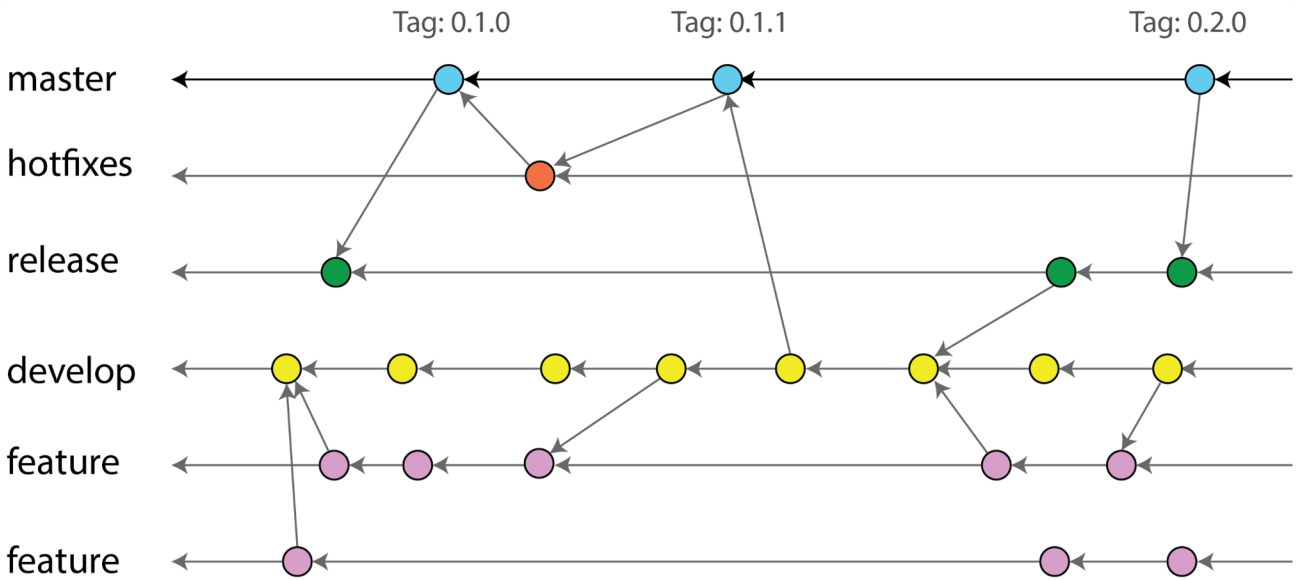
`$ git push origin main`	Update the origin/main branch with my current branch (one time push)
`$ git push -u origin main`	Update the origin/main branch with my current branch and set it as upstream for the current branch.

`git pull`

Incorporates changes from a remote repository into the current branch. In its default mode, git pull is shorthand for git fetch followed by git merge FETCH_HEAD.

`$ git pull`	Pulls code from upstream into the current branch.

Git branching model



- Remove your feature branch after merging your branch to the codebase.
- Always work on a new branch, even if you are adding a couple of lines to the code.
- Even when you are working on a project alone, you and yourself a couple of month later are two different people.

https://nvie.com/posts/a-successful-git-branching-model/
https://docs.github.com/en/get-started/quickstart/github-flow

Git / Github best practices

- Github profile
  - Make sure it is updated.
- Github issues
  - Simple reproducible problem for bugs.
  - User story style issues for feature request.
- Github Discussions
  - For general and long term discussions.
- README file
  - A brief introduction about the project and how to contribute.
- CHANGELOG file
  - Provides a summary of changes for each release
- CITATION file
  - Provides information on how to cite the project.
- Submitting a Pull Request (PR)
  - What does it mean?
  - What is Continuous Integration (CI)
  - What is Work In Progress (WIP)
  - PR comment (Closes #Issue_number)

References

- https://git-scm.com/book/en/v2
- https://swcarpentry.github.io/git-novice/index.html
- https://choosealicense.com/
- https://opensource.org/licenses
- https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002598
- https://nvie.com/posts/a-successful-git-branching-model/
- https://docs.github.com/en/get-started/quickstart/github-flow
- https://www.oreilly.com/library/view/git-version-control/9781789137545/12ad80de-2c0d-43b6-8157-b991084640e3.xhtml
- https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent
- https://git-lfs.github.com/