# On Git – a version control system

## A real-life time machine !

Department of DS and CSE

IIT Palakkad

March 26, 2025

# What is Git ?

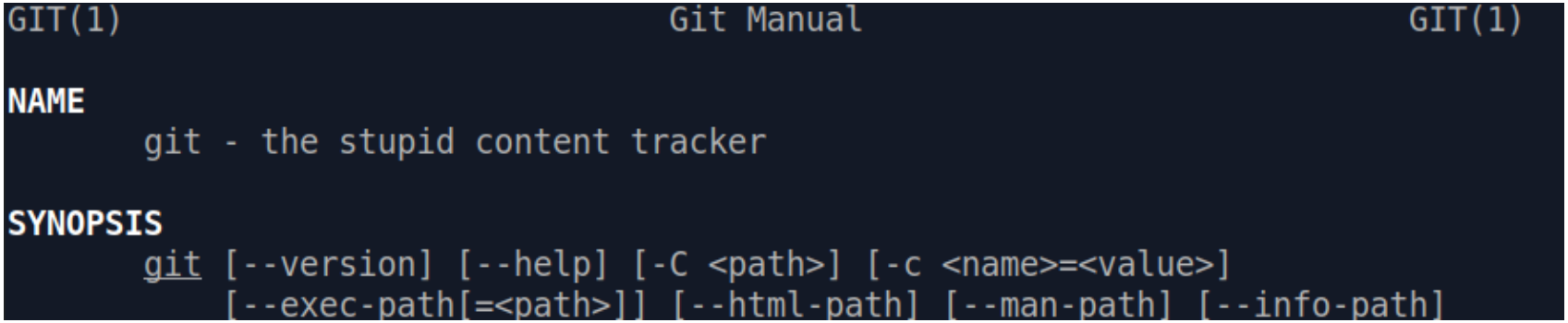


Figure 1: Man page of Git

- Version control system
  - track changes in a personal/collaborative setting
  - recover files, compare changes
  - versioning

# What is Git ?

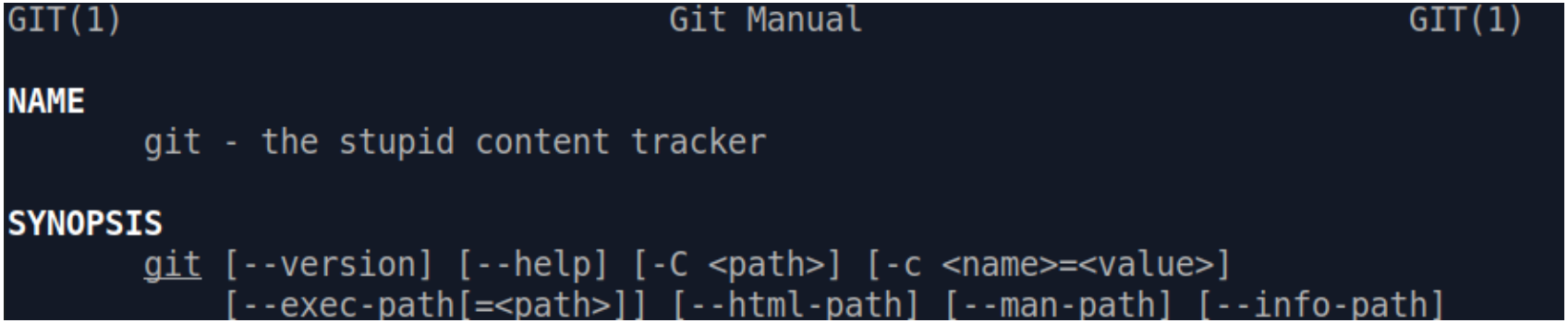


Figure 1: Man page of Git

- Version control system
  - track changes in a personal/collaborative setting
  - recover files, compare changes
  - versioning
- Developed by Linus Torvalds (for Linux Kernel)

# Reasons (people have) for **not** using Git

- Create copies ? Google Drive ? Dropbox ?

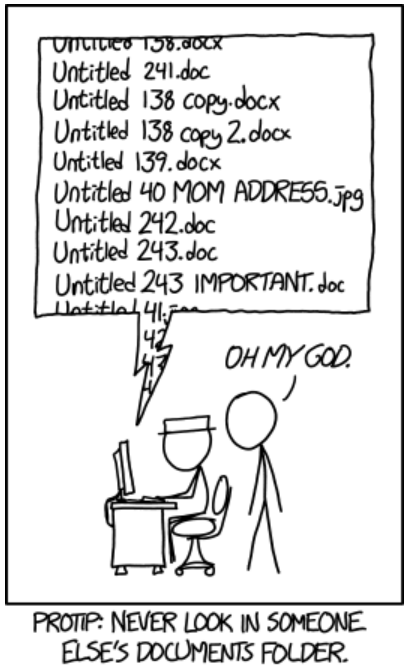


Figure 2: Documents: XKCD

- When do Git work ? When does it not ?

# Reasons (people have) for **not** using Git

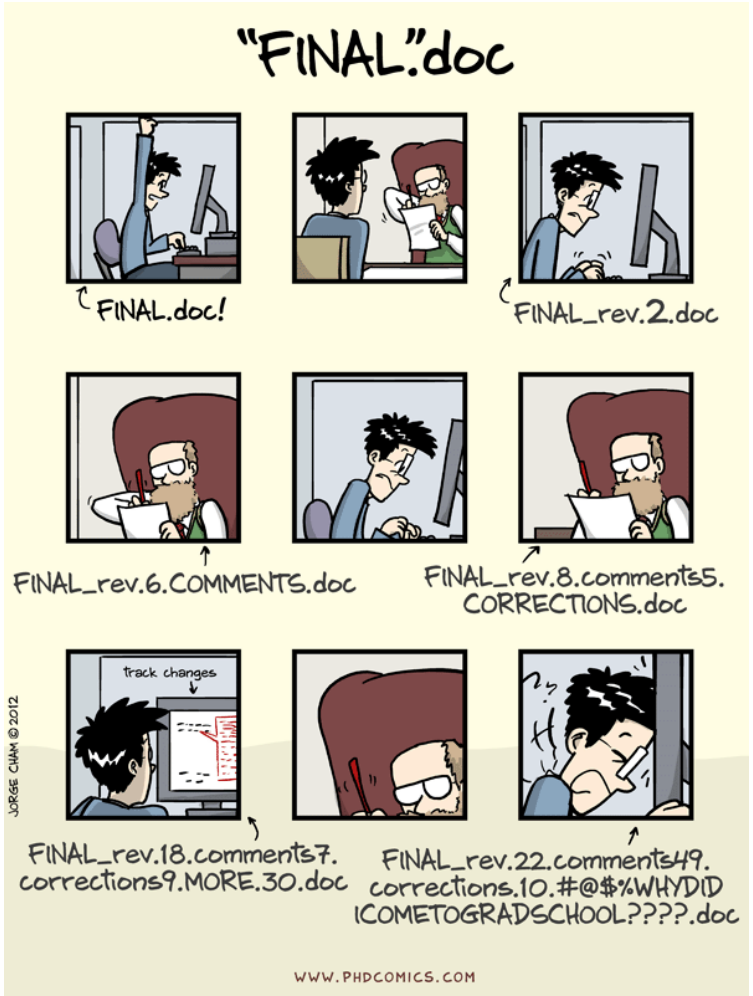


Figure 3: PhD Comics

# More reasons (people have) for **not** using Git

```
$ ls

Specifications20141212.txt
Specifications20141216.txt
Specifications20141212_v2.txt
Specifications20141212_FINAL.txt
Specifications20141212_final_v2.txt
Specifications20141216_final_v2_FINAL.txt
```

# Some cool features of Git

- Maintaining various versions without duplicating.
- Restoration to a stable state. (Similar to Ctrl+Z)
- Try changes without fear of breaking working code.
- Identifying what was the breaking change.

# What next ?

- A mental model of how Git operates
  - **Time travelling machine**
- Plan on Git
  - Introduce key ideas
  - Illustrate use via 5-6 scenarios
- Assignment
  - A Game `oh-my-git`

# Basics of Git

- Git is a tracker. Git operates on a *repository* (repo)
- What is in a repo ? Three types: files that
  - ... are *Untracked*
  - ... have *Changed* (also called *Modified/Working dir*)
  - ... are in *Staging* (also called *Index*)
- *History* - Record maintained by git
- **Adding** and **committing**.
- `HEAD` - pointer to current state
- Commits identified by hash (SHA-1)

# Scenario 0: Make changes to a timeline (`git commit`)

**Commit**

- What ?
  - Record changes made to `git` database.

# Scenario 0: Make changes to a timeline (`git commit`)

**Commit**

- What ?
  - Record changes made to `git` database.
  - (More precise: Record the changes in Staging area to History.)

# Scenario 0: Make changes to a timeline (`git commit`)

**Commit**

- What ?
  - Record changes made to `git` database.
  - (More precise: Record the changes in Staging area to History.)
- When to use ?
  - Take a snap-shot of current work
  - Feel like taking a backup of the whole folder ...

# Scenario 0: Make changes to a timeline (`git commit`)

**Commit**

- What ?
  - Record changes made to `git` database.
  - (More precise: Record the changes in Staging area to History.)
- When to use ?
  - Take a snap-shot of current work
  - Feel like taking a backup of the whole folder ...
- How to use ?
  - `git commit -m "A commit message"`

# Scenario 0: Make changes to a timeline (`git commit`)

**Commit**


Figure 4: Git commit: XKCD

- Too lazy to add ? Commit in one-shot ?
- `git commit -a -m "meaningful commit message"`

# Scenario 1: Create an alternate timeline (`git branch`, `git switch`)

**Branch**

- What ?
  - Create a branch.
- How to use ?
  - `git branch Asgard`
- When to use ?
  - Make changes and experiment without affecting original work.

# Scenario 1: Create an alternate timeline (`git branch`, `git switch`)

- `git branch`: only creates a branch.
- `git switch`: switch to a branch. Usually done after creating a branch.
- Usage: `git switch Asgard`

# Scenario 1: Create an alternate timeline (`git branch`, `git switch`)

- `git branch`: only creates a branch.
- `git switch`: switch to a branch. Usually done after creating a branch.
- Usage: `git switch Asgard`
- List all branches : `git branch -l`
- Delete a branch : `git branch -D Wakanda`

# Scenario 2: Explore the timeline (`git log`)

**Log**

- What ?
  - View log of what all changed.
- When to use ?
  - Want to see the list of changes so far
- How to use ?
  - `git log`

# Scenario 2: Explore the timeline (`git log`)

**Log**

- What ?
  - View log of what all changed.
- When to use ?
  - Want to see the list of changes so far
- How to use ?
  - `git log`
- How to get more out of `log` ?
  - For graph view - `git log --oneline --graph --color`

# Scenario 3: Travel to the past (and come back !) (`git checkout`)

**Checkout**

- What ?
  - Move to an existing commit, identified by its hash value.
- When to use ?
  - To revisit an earlier version.
  - (Git commit messages can help)
- How to use ?
  - `git checkout 12ab20`
- **Demo**

# Scenario 3: Travel to the past (and come back !) (`git checkout`)

**Checkout**

- What ?
  - Move to an existing commit, identified by its hash value.
- When to use ?
  - To revisit an earlier version.
  - (Git commit messages can help)
- How to use ?
  - `git checkout 12ab20`
- **Demo**
- To get back, do: `git checkout -`

# Scenario 4: Where am I ? (`git status`)

**Status**

- What ?
  - Tell the current state of the repo
  - `git status` is your friend !
- When to use ?
  - I don't know what is happening !
  - Somebody asks you to fix their repo !
- How to use ?
  - `git status`

# Summary so far

| **Function** | **Command** |
|---|---|
| Commit changes | `git commit` |
| Create branch, switch | `git branch`, `git switch` |
| Checkout another commit | `git checkout` |
| Current status | `git status` |
| Full status | `git log` |
| To track changes | `git add` |

# Summary so far (Coming next)

| **Function** | **Command** | **Approx. Opposite** |
|---|---|---|
| Merge branches | `git merge` | `git branch` |
| Undo changes | `git reset`<br>`git restore`<br>`git revert` | `git add`<br>`git commit` |

# Summary so far (Coming next)

| **Function** | **Command** | **Approx. Opposite** |
|---|---|---|
| Merge branches | `git merge` | `git branch` |
| Undo changes | `git reset`<br>`git restore`<br>`git revert` | `git add`<br>`git commit` |
| Remote operations | `git fetch`<br>`git pull`<br>`git push` | |

- Repository – (1) origin and (2) **remote**

# Scenario 5: Merge two timelines (`git merge`)

**Merge**

- What ?
  - Merges `HEAD` with another commit
- When to use ?
  - Combine changes made in two different branches
- How to use ?
  - `git merge Asgard`
- **Demo**

# Scenario 6: Undo changes to a timeline (`git restore`, `git revert`)

**Restore**

- What ?
  - Restore files in working directory.
- When to use ?
  - Edited a file. Did not commit. Want to restore it to last commit.
  - Changes made will not be committed.
- How to use ?
  - `git restore password.txt`
- **Demo**

# Scenario 6: Undo changes to a timeline (`git restore`, `git revert`)

**Revert**

- What ?
  - Undo all the changes done in a commit.
- When to use ?
  - Found a commit creating problems. Want to undo all the changes.
  - Changes made will be committed.
- How to use ?
  - `git revert 0e70093`
- **Demo**

# Scenario 7: Reset back current timeline (`git reset`)

**Reset**

- What ?
  - Move the `HEAD` to a previous commit, while abandoning any changes
- When to use ?
  - Screwed up the code making changes.
  - (Want to unroll the changes)
- How to use ?
  - `git reset HEAD^`
- **Demo**

# Scenario 8: Import changes from the remote (`git fetch`)

**Fetch**

- What ?
  - Only fetch the changes made in remote (does not merge the changes)
- When to use ?
  - Get changes from commits made by other users
- How to use ?
  - `git fetch`
- `git pull` = `git fetch` + `git merge`

# Scenario 9: Makes changes to the remote (`git push`)

**Push**

- What ?
  - Push changes made locally to remote
- When to use ?
  - Share changes made to all other users
- How to use ?
  - `git push`
- **Demo**

# Scenario 10: Create / Copy repository (`git init`, `git clone`)

**Initialize**

- What ?
  - Create an empty repo locally
- When to use ?
  - Want to track changes for you alone
- How ?
  - `git init`

# Scenario 10: Create / Copy repository (`git init`, `git clone`)

**Clone**

- What ?
  - Create copy of an existing repo
- When to use ?
  - Want to track changes with fellow developers.
- How ?
  - `git clone https://github.com/torvalds/linux.git`

# Overall Summary

| Function | Command |
|---|---|
| Commit changes | `git commit` |
| Create branch, switch | `git branch`, `git switch` |
| Checkout another commit | `git checkout` |
| Current status | `git status` |
| Full status | `git log` |
| Track changes | `git add` |

# Overall Summary

| **Function** | **Command** | **Approx. Opposite** |
|---|---|---|
| Merge branches | `git merge` | `git branch` |
| Undo changes | `git reset`<br>`git restore`<br>`git revert` | `git add`<br>`git commit` |

# Overall Summary

| **Function** | **Command** | **Approx. Opposite** |
|---|---|---|
| Merge branches | `git merge` | `git branch` |
| Undo changes | `git reset`<br>`git restore`<br>`git revert` | `git add`<br>`git commit` |
| Remote operations | `git fetch`<br>`git pull`<br>`git push` | |

# Local versus remote

- Commands to manage remote
  - `git fetch`
  - `git push`
  - `git pull` = `git fetch` + `git merge`

# Hmm !

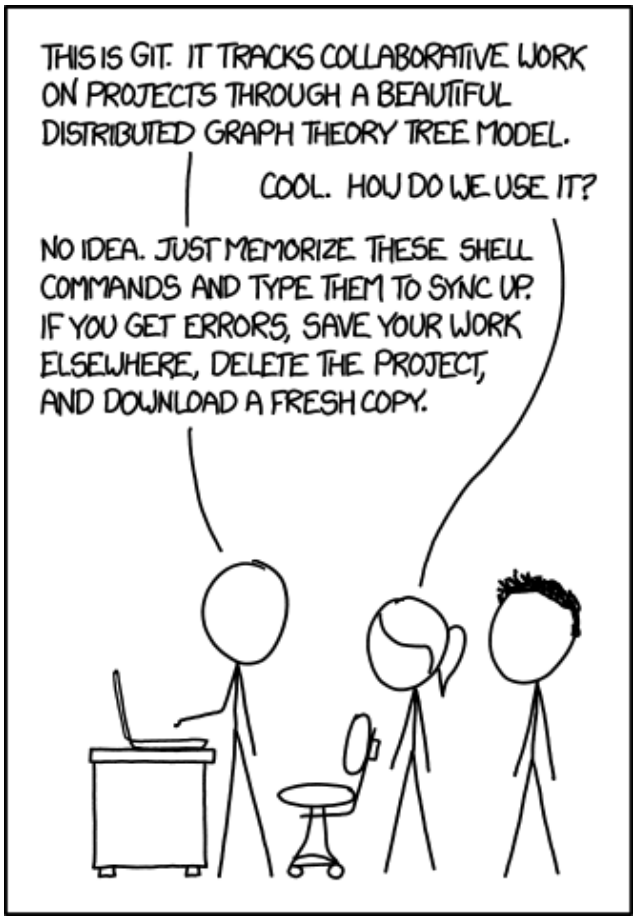


Figure 5: Git: XKCD Comics

# Basic Git Usage

- git config
- `.gitignore` file

# Basic Git Usage

- git config
- `.gitignore` file
- git add, git rm, git mv
- **Basic work: `git add`, `git commit`, `git push` loop**
- git diff (between current and an earlier commit)
- git tag (tags names instead of hashes for commits)

# References


Figure 6: Pointers: XKCD

- `man gittutorial`, `man giteveryday`
- Git at git-scm, Git sim - visual operations repo
- XKCD and PhD Comics for comic strips

# Colophon: Why the name Git ?

```
GIT - the stupid content tracker

"git" can mean anything, depending on your mood.

 - random three-letter combination that is pronounceable, and not
   actually used by any common UNIX command.  The fact that it is a
   mispronunciation of "get" may or may not be relevant.
 - stupid. contemptible and despicable. simple. Take your pick from the
   dictionary of slang.
 - "global information tracker": you're in a good mood, and it actually
   works for you. Angels sing, and a light suddenly fills the room.
 - "goddamn idiotic truckload of sh*t": when it breaks

This is a stupid (but extremely fast) directory content manager.  It
doesn't do a whole lot, but what it _does_ do is track directory
contents efficiently.
```

Figure 7: Initial commit by Linus