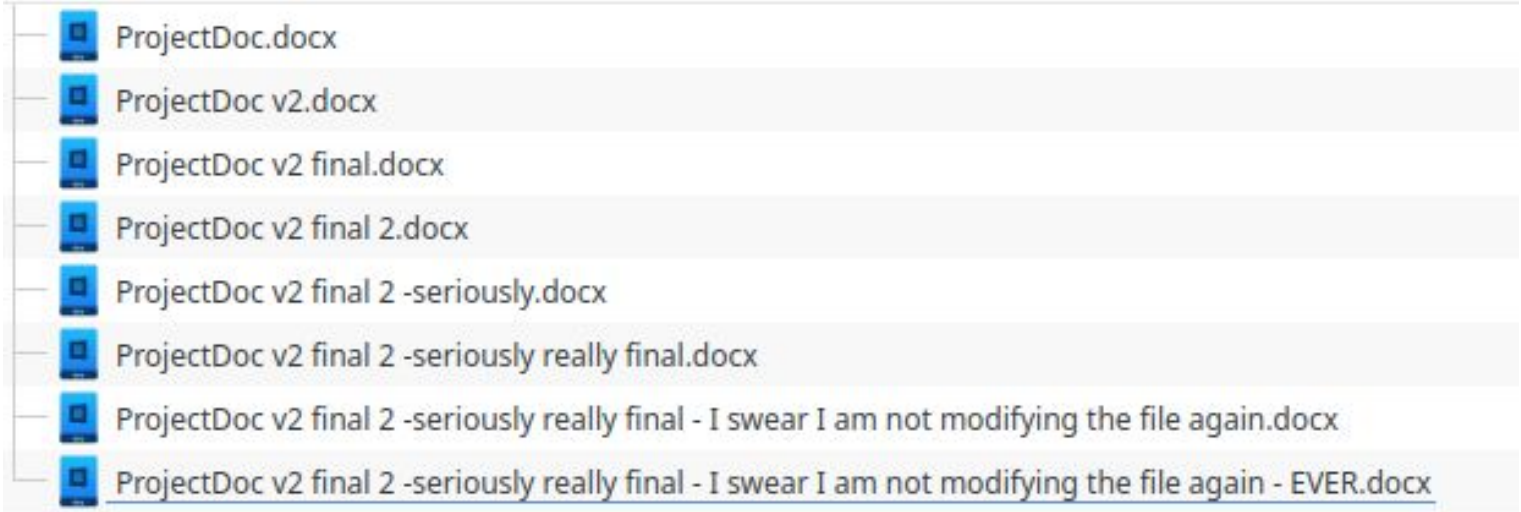
Practical Git

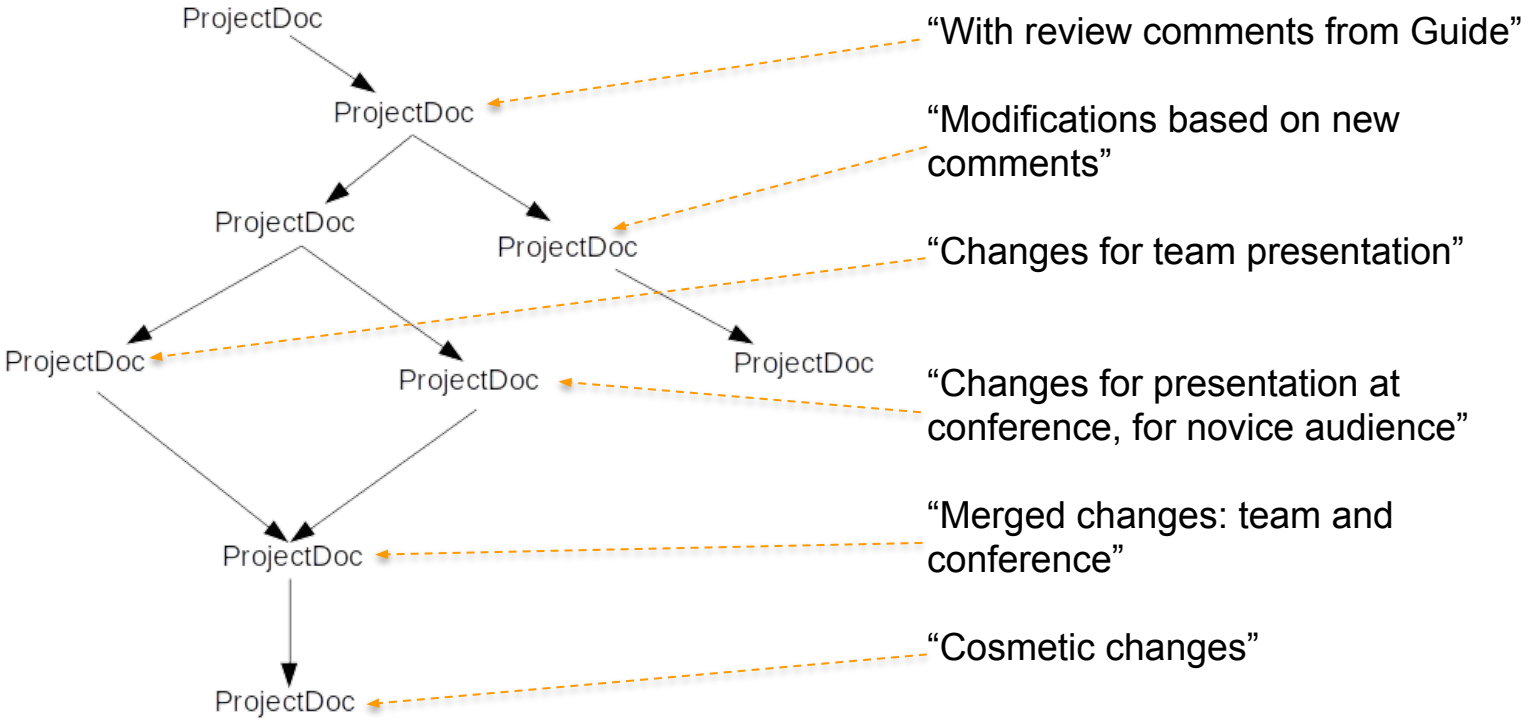
Introduction (And Beyond)

@nikhilwanpal | NikhilWanpal



Version Control

(The non-developer way)



Version Control

(The developer way)

Version Control Systems

Types:

1. Local
2. Centralized (CVCS): svn, cvs
3. Distributed (DVCS): git, mercurial

Once upon a time..

- There was a Linus
- Then came Linux
- Then came the interest of the community and contributor boom
- Contributors distributed through time and space.
- The central person would become overworked, loaded.
- Need for distributed management.
- (skipping some events..) Birth of Git: To handle far complex, large and distributed teams. (than us)
- ..and then, came the interest of the community and contributor boom… (git uses git to version control git!)

CVCS Vs DVCS

1. Central Server, that manages the truth.
2. Clients do checkout snapshots.
3. Weak clients
4. Server needs tending to: backups, maintenance etc.
5. File locking / conflict handling.
6. Constant connection required.
7. Defined workflow.
8. Restricts free development of open source projects.

1. Distributed, no single location of truth.
2. All clients are mirrors; servers; truth.
3. Fully functional clients
4. Self maintaining, recoverable from mirrors; (of course should have backups.)
5. No Locking; conflicts are less frequent.
6. Connection required only when sharing.
7. Highly flexible workflows possible. With subteams and sharing and merging before final publishing.
8. Promotes Open Source development.

Understanding Git (coming from a world of CVCS)

- It is distributed: Things appear to be two step.
- Think of ‘torrents’, P2P networks. (It’s not a P2P and won’t help you download GOT!!)
- Like installing your own SVN server that can communicate with other SVNs.
- Branching is dirt cheap; not a task.
- Merging is easy; not an activity. (If you know what you are merging!)
- It’s not difficult, just different. (unless our glass is full)

Git Hosting

- GitHub Vs GitLab Vs BitBucket
- Git : GitHub = water : packaged drinking water
- All hosting providers add proprietary features to git, ex: pull request.
- Hosting providers do, can provide other VCS as well.

- Git does not *need* hosting, or server or background process
- Git can work with a shared folder as remote
- Hosting makes corporate workflows easy

Your Git

- Install Git

- Setup

```
git config --global user.name "Nikhil Wanpal"
git config --global user.email "nikhil@dontwasteyourtimereading.com"
```

Your Git, Your First Repository

In a new directory: practical-git/1/

- git init
- Create file, git add, git commit -m
- Modify file, git add, git commit -m
- git log
- Modify file, git add, git commit --amend
- Modify file, git add
- git reset

Your Git, Your First Repository with a server

In a directory: practical-git/central-repo.git/

- `git init --bare`

In a directory: practical-git/personal-repo/

- `git clone ../central-repo.git/ .`
- git add
- git commit -m
- git push

Your Git, Your First Repository with a server and a dual personality

In a directory: practical-git/central-repo.git/

- `git init --bare`

In a directory: practical-git/personal-repo/

- `git clone ../central-repo.git/ .`
- Create file, git add, git commit -m
- git push

Your Git, Your First Repository with a server and a dual personality (v1)

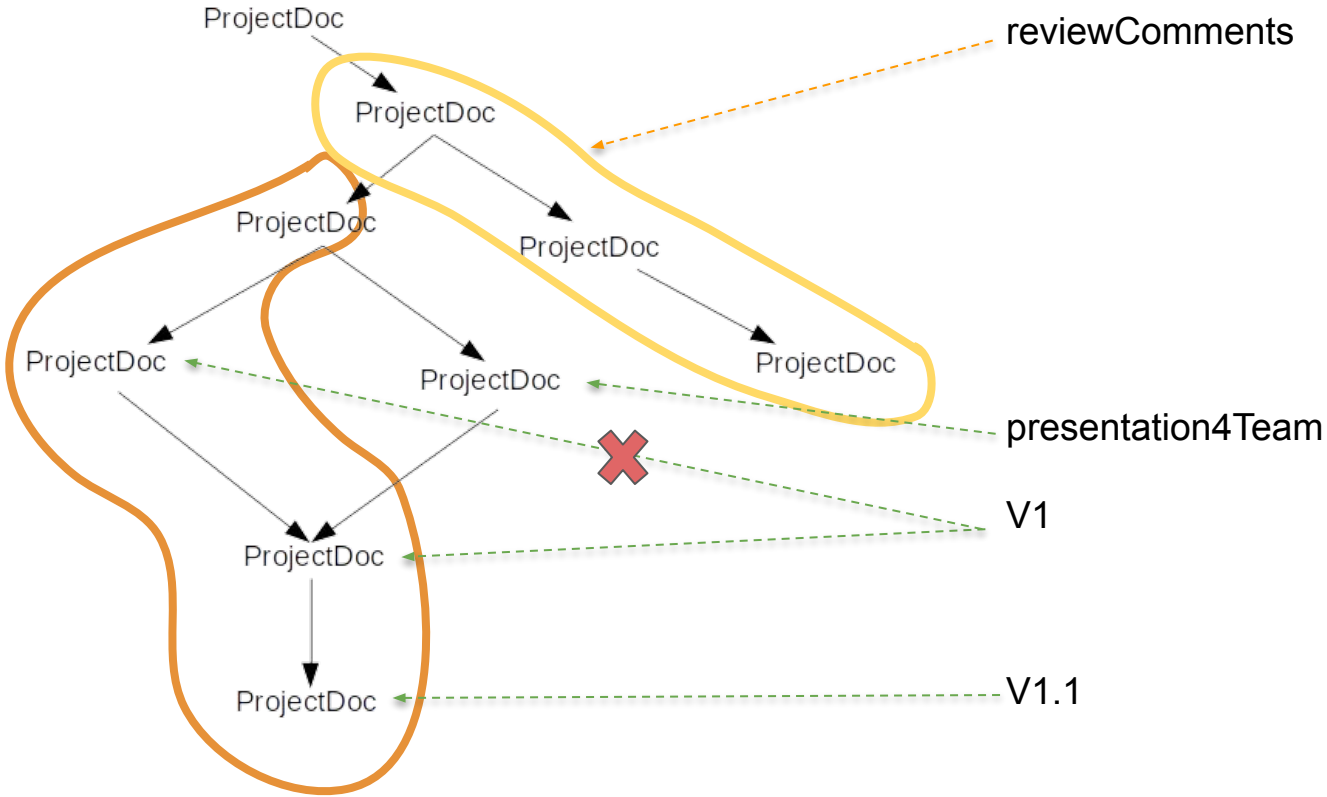
In a directory: practical-git/colleague-repo/

- `git clone ../central-repo.git/ .`
- Surprise!
- Create file, git add, git commit -m
- git push

In a directory: practical-git/colleague-repo/

- git pull

Branches and Tags



V1.1 Branches

- Threads of development
- String of thought
- Series of changes with similar purpose
- A diversion

```
git branch myFirstBranch
```

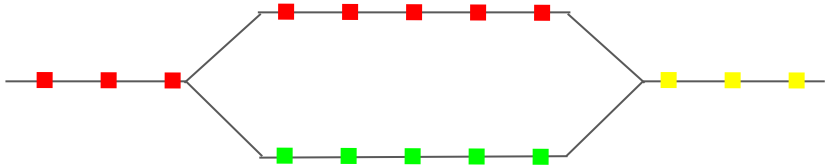
- Create a file, git add, git commit -m, git push

```
git checkout master
```

V1.2 Merges

- Tie the threads together
- Bring together different thoughts and ideas
- Achieve the purpose of your branch

```
git merge myFirstBranch
```



- Git's intelligent merge

```
git checkout myFirstBranch, modify line 1, push
```

```
git checkout master, modify line 2, merge
```

V1.3 Conflicts

- What happens when two different lines of thought try to merge?
- Branches need a judge!
- You!

```
git checkout myFirstBranch, modify line 2, push

git checkout master, modify line 2, merge!
```

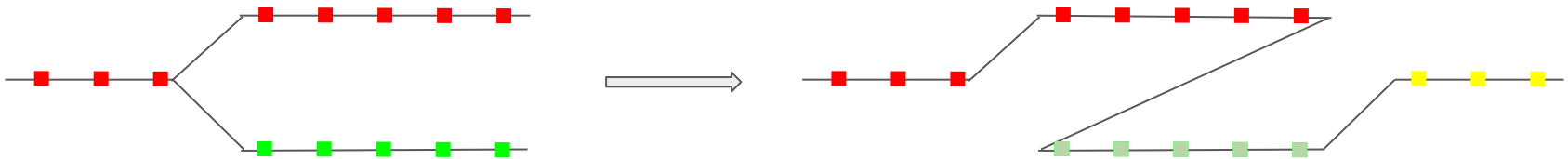
- Now try that across repositories, you have 2.

V1.4 Rebase

- Rewriting history

(after same steps as a conflict..)

```
git pull origin master --rebase
```



What is a:

- Git commits storage: a rooted, **D**irected **A**cyclic **G**raph of patches.
- **Patch**: the delta between two commits. (or more)
- **Staging/Index**: Selecting relevant changes for commit.
- **Commit**: The delta store in git filesystem with name as the SHA.
- Branch **HEAD**: A pointer in the graph for ease of access.
- **Branch**: The path from root to branch head.
- **Stash**: Stash aside the state for now.
- **HEAD**: a variable, a pointer to current pointer of the current branch.
- **Detached HEAD**: state of repository when a commit is checked-out, which is not pointed to by any of the HEADs

What is a:

- **Merge**: Delta from the ‘common ancestor’ added together.
- **Fast - Forward** Merge: Pointer updated to the latest head.
- **Merge commit**: the commit that identifies a merge.
- **Rebase**: Rewrite the history to change the branching point, and reapplying the changes over. No more the same commits. (creating new history is that easy..!)
- **Reset**: undo, hard vs soft.
- **Remote**: The different repositories, tracked branches, not same branches.
- **Pull**: fetch + merge (rebase!)

Git Folders:

- HEAD: pointer to current branch’s head
- index: staging info
- refs: commit objects. Basically branch data.
- objects: blobs of files and tree objects.

Git Objects:

- blobs: or git objects contain the contents of checked files. Key-value file storage.
- trees: pointers to blobs by filenames and other trees.
- Commit: top level tree, user, additional info regarding commit, message etc.
- Packfiles and git gc | auto gc.

To Branch or not to branch..

What is a branch? (There is no such thing as a branch! It's the path from the head to root, traversed through 'parent' pointers.)

Git References:

- Branches
- Tags
- HEAD
- Remotes

Are you coming back to it? Then you need a branch!

Git Flow

- A branching model suggested by Vincent Driessen in his blog.
- develop and master: The only long lived branches.
- Features, Releases and Hot-fixes
- Convention: feature/; release/ and hotfix/
- Life-cycle of:
  - Feature: develop → develop
  - Release: develop → (master + tag | develop)
  - HotFix: master → (develop + master(tag?))

Git Shortcuts

Bash / shell aliases:

- `alias gs='git status '`
- `alias ga='git add '`
- `alias gc='git commit'`
- `alias gb='git branch '`
- `alias go='git checkout '`

Git aliases:

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

Git Best Practices

- Commit often (every 30 mins), push once. Clean-up before push.
- Prefer to code on new branch locally, never push such branches. Share among developers but not to central.
- One change per commit. Not more.
- Describe the commit well.
- Consider rebase before push or pull, follow up with a --no-ff commit.
- Don’t break the development tree.
- Review merges. Build and fix post a merge before push.
- Avoid force-delete (-D) when deleting branches.
- Consider using shortcuts/aliases.
- **NEVER rebase pushed commits.**

}