Git

User Manual

Content

Git - Introduction

- Fast distributed ‘Revision Control System’
- ‘Git’ has no special meaning of full form but feel free to get some ideas
- Getting help on git commands (e.g. `clone`):
  - `man git-clone`
  - `git help clone`
- Throughout this lesson, we will be using my simple ‘art_gallery’ website project repo’s copy ‘art_gallery_git_exercise’ to practise git commands

Repositories and Branches

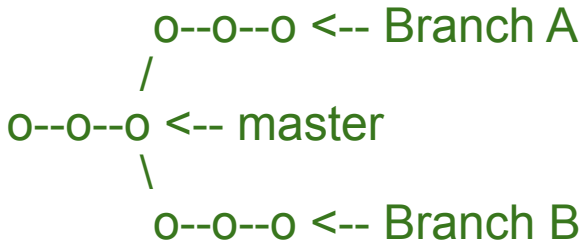
- How to get a git repo?
  - ➢ Cloning existing repo on GitHub
    - `git clone url_to_repo`
      - E.g. `$ git clone git://git.kernel.org/pub/scm/git/git.git`
  - ➢ Adding a local project to GitHub
    - `git init` // creates a .git dir (contains all history of the project)
    - `echo “# about the project” >> README.md` // write to a readme file
    - `git add *` // stage all files
    - `git commit -m “first commit”` // commit all staged files
    - `git remote add origin git://_repo_url` // add repo url
    - `git push -u origin master` // upload project to remote repo

Repositories and Branches

- How to check out a different version of a project?
  - commit: a snapshot of a version of a project
  - History of the project is hence a collection of **commits**
  - History is not always a straight line from oldest to newest commit/snapshot
    - It may have **branches** which my merge and diverge
    - The latest commit on each of a repo's branch is called a **head**
    - Git can track development on multiple branches by keeping a list of **heads**
  - `git branch` // shows list of branches (default 'master or main')
  - `git tag -l` // also shows history
    - While **tags** always point to a given version in history, **heads** are expected to advance with the project
- `git show` // show details of the latest commit
  - Every commit has a 40-hexdigit **id** called *object name* or *SHA1-id*
- Each commit except the first one has 1 or more **parent** commits
  - See details of the commits ancestry with **`gitk`** command

Repositories and Branches

- History diagrams



- Time goes from left to right
- Each ‘o’ is a commit
- The 3 commits on the first line is called **branch** A
- The last commit of the three in branch A is called the **head** ‘A’

Repositories and Branches

- Manipulating Branches
  - `git branch` // list all branches
  - `git branch <name>` // creates a branch with the given name
  - `git branch -d <name>` // delete the branch with given name
    - If the branch has not fully merged with the upstream branch or contained in the current branch, the command fails
  - `git branch -D <name>` // delete given branch regardless
  - `git switch <branch>` // make the current branch <branch>, update working directory to reflect the version given by <branch>
  - `git switch` usually expects the branch HEAD to switch to
    - But if provided with a `--detach` flag, will accept to switch to any arbitrary commit
    - E.g. you can switch to a commit referenced by a tag e.g. `'v2.6.1'`
    - `git switch --detach v2.6.1`
      - The HEAD refers to the SHA1-id of the commit and not the branch
      - `git branch` // `* (detached from v2.6.17)` (shows you are no longer on a branch)
      - In this case we call the HEAD is detached
      - An easy way to checkout a specific version without having to create a new named branch

Repositories and Branches

- Manipulating Branches from a Remote Repo
  - The ‘master’ branch in your machine is a copy of the HEAD in the repo you cloned from
  - You might have added branches to the master
  - Similarly, other people who cloned the repo may have added branches to it
  - You can see those ‘remote branches’ using `git branch -r`
- Git Fetch
  - Since you cloned a repo, you might have a made your own few commits, few branches
  - You might also wish to look for updates on the repo
  - `git fetch` // updates all remote-tracking branches to the latest version found in the repo
    - It won’t touch your local branches, not even your local ‘master’

Exploring Git History

- Git is a tool to store history of a collection of files
- It does so by storing snapshots of contents of the file hierarchy with commits
- Commits show the relationships of these snapshots
- Git provides extremely fast tools to examine the history of a project
- TO BE CONTINUED...

Developing with Git

- Telling Git your name:
  - `git config --global user.name “Kamal Thapa”`
  - `git config --global user.email “thapakml@gmail.com”`
  - This will add a .gitconfig file in the home dir (~/.gitconfig)
- Creating a new repository
  - From scratch
    - `mkdir my_project`
    - `cd my_project`
    - `git init`
  - If you have some initial content e.g. old_project
    - `cd old_project`
    - `git init`
    - `git add .` // include everything below for staging before first commit
    - `git commit`

Developing with Git

- Git Commit
  - Step 1: make the changes you want to the project
  - Step 2: tell git about your changes (`git add`)
  - Step 3: create the commit
- Git maintains a snapshot of the tree's content in a special staging area called 'the index'
  - `git add filename` // add file to the index
  - `git rm filename` // remove file from the index
- Commands useful to track what you are about to commit
  - $ `git diff --cached` # difference between HEAD and the index; what would be committed if you ran "commit" now.
  - $ `git diff` # difference between the index file and your working directory; changes that would not be included if you ran "commit" now.
  - $ `git diff HEAD` # difference between HEAD and working tree; what would be committed if you ran "commit -a" now.
  - $ **`git status`** # a brief per-file summary of the above.

Developing with Git

- Creating good commit messages
  - < 50 char to summarize the change (Git treats the first line as title)
  - A new line char, then the details
- Ignoring files in Git
  - You might have some files you do not want git to track and hence add, commit, push etc.
    - E.g. node_modules, pip packages
  - Add those file names in a .gitignore file
    - '#' for comments, *.txt will ignore all .txt files etc.
- Merging
  - `git merge <branch_name>` // merges the given branch to current branch
  - Combines the <branch_name> and the changes made until the latest commit current working
  - Commit your changes before merge (if you have uncommitted changes touching the same files in the branches you want to merge, Git won't proceed the merge)
  - If you do not want to commit the changes, `git stash` to take the changes away while merging so you can re-apply the changes after