

Version Control with Git

Linux Users Group
UT Arlington

Rohit Rawat
rohitrawat@gmail.com

Need for Version Control

- Better than manually storing backups of older versions
- Easier to keep everyone updated on a large project
 - Allows concurrent development by multiple developers and safely merges those changes
 - Accountability for all changes
 - Fine tuned access control

Version Control Systems

- Examples: CVS, SVN, Git, Mercurial, and other proprietary tools
- Git
 - Created by Linus Trovalds
 - Distributed VCS
 - CVS and SVN use centralized model
 - <http://git-svn.com>
 - Free and open source
 - Cross platform

Distributed VCS

- Each user has a copy of the whole repository, i.e. all the code and all the revisions.
- Advantages:
 - Safer due to redundancy
 - Fast access, can also work offline
 - No central server required
 - Less hesitation in branching and making commits
- Disadvantages:
 - Commits are local/delayed
 - First time download is slow as you also download the complete project history

Distributed VCS

- Each user has a copy of the whole repository, i.e. all the code and all the revisions.
- Advantages:
 - Safer due to redundancy
 - Fast access, can also work offline
 - No central server required
 - Less hesitation in branching and making commits
- Disadvantages:
 - Commits are local/delayed
 - First time download is slow as you also download all project history

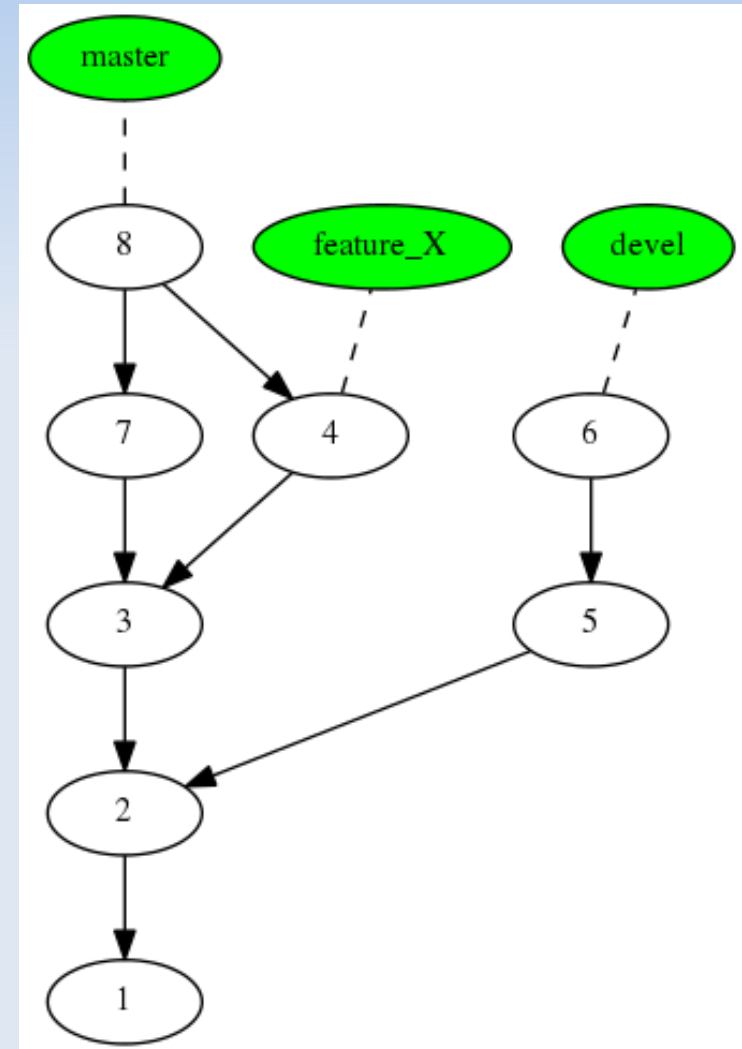
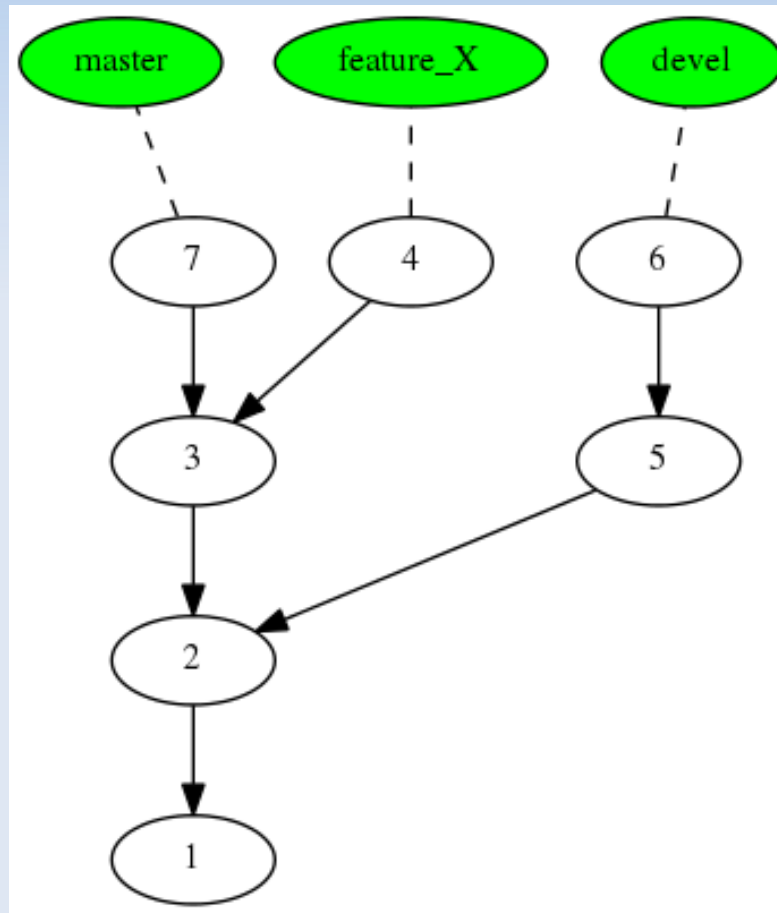
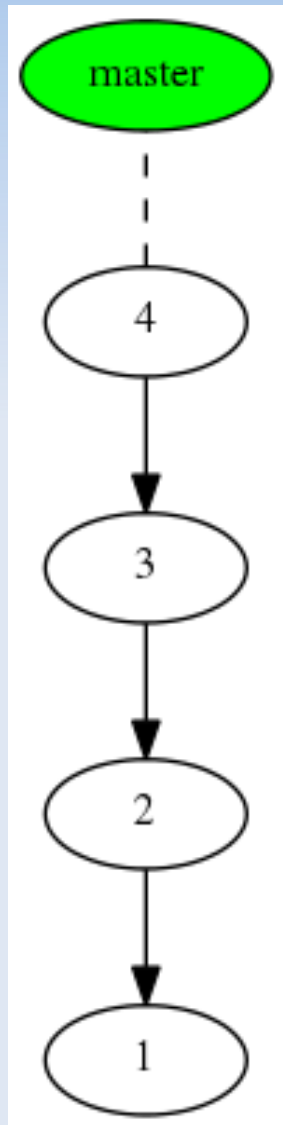
Is Git hard to set up?

- Git easily installs on Windows/Linux/Mac
 - You can create a local repository and start working with it immediately
- You can synchronize your repository to an online service to collaborate with others, or just to have a remote backup
- Free online hosting is available
 - GitHub, Bitbucket, SourceForge, Assembla
 - You can also set up your own hosting server

VCS Terminology

- Repository – a location where all your code revisions and history is stored
- Commit – 1. v. The act of recording changes into the repository. 2. n. A snapshot of your code created during a commit
- Branch – A sequence of commits leading to a particular code state.
 - Different branches lead to different versions of code
 - Convention is to have a "master" branch for your stable code, separate branches for development work

Repositories and branches



More terminology

- Working copy – the code present on your machine, with changes that may or may not have been committed
- Staging area – Git collects all files submitted with the 'git add' command in a staging area till you finally decide to commit them
- Unversioned files – Files which are not put under version control, like temporary files, binaries.

Using Git

- Linux – install **git** package
- Windows – install **msysgit**
- Graphical tools are also available – GitGUI, TortoiseGit
- Configure git for first time use:
 - `git config --global user.name "Rohit"`
`git config --global user.email "rohit.rawat@mavs.uta.edu"`
* Your username and email are logged with each commit.

Two workflows

- Create a new local repository and start adding files to it
 - You link it to an online host like GitHub now or later
- Clone an existing repository from a git server
 - This creates a copy of the repository on your machine and links it to that source

Initialize Git repository

- Git repositories reside with the code

Project_folder/

code1.cpp, code2.cpp, subfolders, '.git' folder

- Initialize git repository, creates '.git' folder:

git init

- Check status of the repository:

git status

Add files to version control

- You need to select which files are to be kept under git

```
git add *.cpp  
git status
```

- Files are added to a staging area. They have not been committed yet and no permanent changes have been made to the repository.
- Make the first commit

```
git commit -m "First import of code into Git!"  
git status
```

Making changes

- Modify a file, then check git status. It should tell you the file has been modified, but not staged.
- If you change multiple files, git does not assume you want to record all those changes in the next commit.
- You have to manually add the changed files to the staging area with git add.
- Or you can commit with the -a option:
`git commit -a -m "Second commit!"`

Comparing versions

- Git lets you compare commits with other commits or your working copy (w.c.).
- Compares the last commit with w.c.:
`git diff`
- HEAD is a label pointing to the the commit responsible for the w.c. To compare w.c. with the version 1 commit behind HEAD:
`git diff HEAD~1`
- To diff with two specific commits, use their SHA code
`git diff sha1 sha2`
- If you manually staged files for commit with `git add`, they don't show up in the diff without the `--cached` switch.

Creating a branch

- Do you need a branch if you are the sole developer?
 - If you are working on more than one feature at the same time, you should do that on separate branches.
 - If you are experimenting with stuff that you are not sure you want to keep, do it on a branch – the master branch will stay clean.
 - Since there are no conflicts with other developers, merging will be fast and easy.

Creating a branch

- Create a branch:
`git branch branch_name`
- List the active branch:
`git branch`
 - The active branch will have an *.
- Switch to the new branch:
`git checkout branch_name`
- Shortcut: `git checkout -b branch_name`
- All further commits go on the active branch.
Make changes and commit them to the branch

Creating a branch

- Create a branch:
`git branch branch_name`
- List the active branch:
`git branch`
 - The active branch will have an *.
- Switch to the new branch:
`git checkout branch_name`
- Shortcut: `git checkout -b branch_name`
- All further commits go on the active branch.

Merge a branch back into master

- Switch to the master branch
git checkout master
- Merge the branch branch_name
git merge branch_name
- If there are any conflicts during the merge, git highlights them in the file using markers ">>>". You can manually open the files and fix the conflicts.
- You can delete the branch to keep things clean
git branch -d branch_name

Remote repositories

- Create a free account on Github
- Follow instructions to create a new repository
- Follow instructions to push an existing repository from the command line:

```
git remote add origin https://github.com/rohitrawat/Demo22.git  
git push -u origin master
```

Remote repositories

- A remote is like an alias for a remote copy of the repository.
- Multiple remotes may be added.
- In this case, we added a remote at Github, which was empty.
- When we pushed out changes, the remote repo got updated with our code.
- When you clone a repo, that source is automatically added as a remote called 'origin'

Add a second developer

- The new developer also creates a Github account, and configures his git installation with the correct email address.
- The original developer logs into Github and adds him under Admin->Collaborators
- The new developer clones the repository:
git clone <https://url/of/the/repository>
 - Clone automatically initializes a new local repo, adds the url as the 'origin', and pulls the code.
 - You may save your public key to your github account to avoid authenticating every time.

Back and forth

- Users may now commit changes to their local repositories.

```
vi code.cpp  
git commit -a
```

- To sync, they would first 'push' their commits to the remote.

```
git push origin master
```

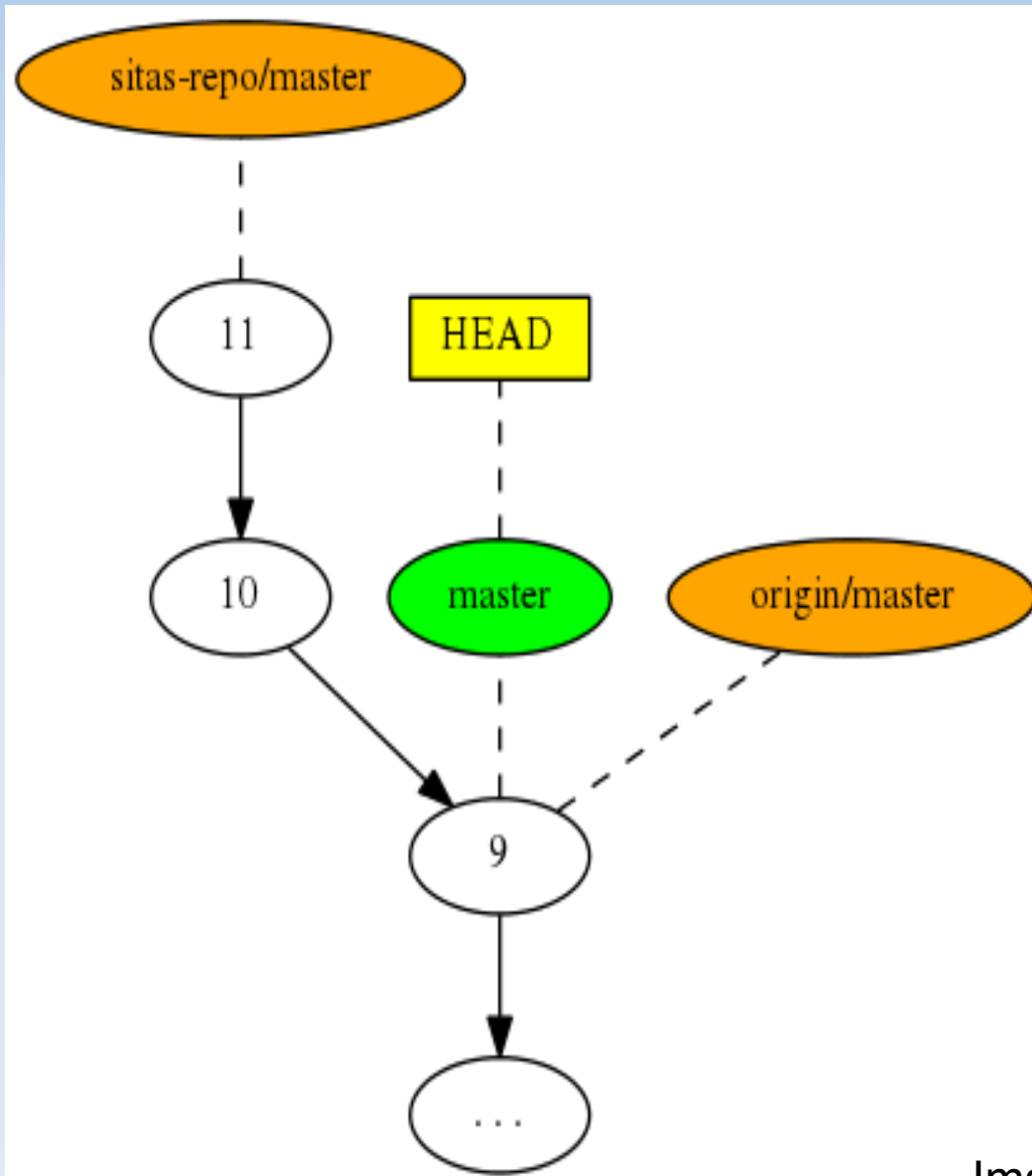
- The other user would then 'pull' from the remote.

```
git pull origin master
```

Git fetch and pull

- Git fetch downloads the latest copy of the remote repository to your machine, but does not affect any code sitting on it.
- Remote branches are different from local branches even if they have the same name!
- You may now diff it with your working copy to see how they differ, and then merge them if you like.
- Git pull does the fetch and merge together.

Different remotes and branches



Here there are two
remotes:
origin and sitas-repo

The local repo and origin
are in sync, i.e. their
master branch is the same.
sitas-repo/master is at a
different commit.

You would want to pull
sitas-repo.

Your own remote repository

- "remote" can refer to any storage location where a repository can be created. It could be a folder on a backup drive.
- You can create an empty shared repo on a shared folder on a Linux server:

```
git init --bare --shared
```

 - This is what Github gives when creating a new repo
- You can give certain users read/write permissions to the folder.
- They may set up a remote or clone that location using the format `user@server:/path/to/shared/repo`

With tools

- You may set up a git server with very fine tuned access controls using a tool like Gitolite or Gitis

References

- [1] <http://sitaramc.github.com/gcs/>

The End