Extra

Git - Version Control System

Lecturer: Duc Dung Nguyen, PhD.
Contact: nddung@hcmut.edu.vn

Faculty of Computer Science and Engineering
Hochiminh city University of Technology

Contents

What is Git

What is Git?

- Created by Linus Torvalds, **creator** of **Linux**, in 2005
  - Came out of Linux development community
  - Designed to do version control on Linux kernel
- Goals of Git
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects efficiently

Git resources

- Git website: `http://git-scm.com/`
  - Free on-line book: `http://git-scm.com/book`
  - Reference page for Git: `http://gitref.org/index.html`
  - Git tutorial: `http://schacon.github.com/git/gittutorial.html`
  - Git for Computer Scientists:
    `http://eagain.net/articles/git-for-computer-scientists/`
- Help at command line: (where command can be 'config', 'commit', etc.)

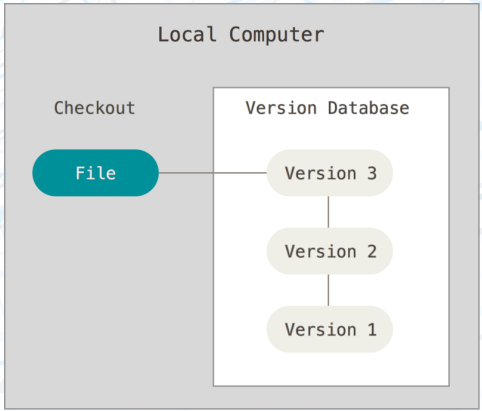
```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

Version Control System (VCS)

- A system that records changes to a file or set of files over time
- You can always **rollback** to some specific point in the development process
- **Compare changes** over time
- See who modified things that might cause a problem
- Who introduced an issue, when, etc.
- A safety check with a very little overhead

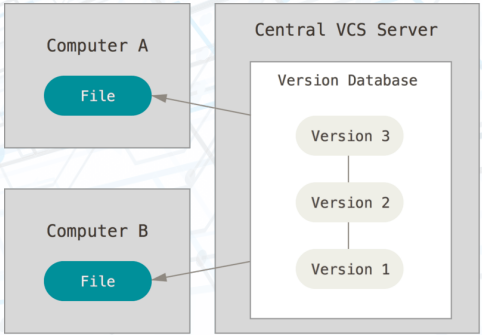
Local VCS

- Record changes, **not copy**
- A local VCS has a simple database to keep all the changes to files under version control.



Centralized VCS

- Issue of local VCS: **collaboration**!
- Centralized VCS: CVS, Subversion, Perforce, etc.
- Clients can check out files from central place.

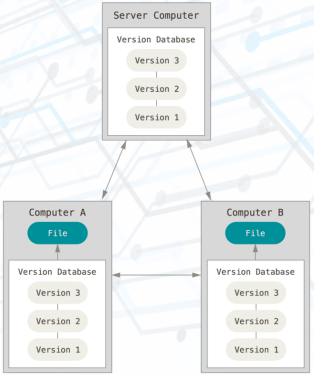


Centralized VCS

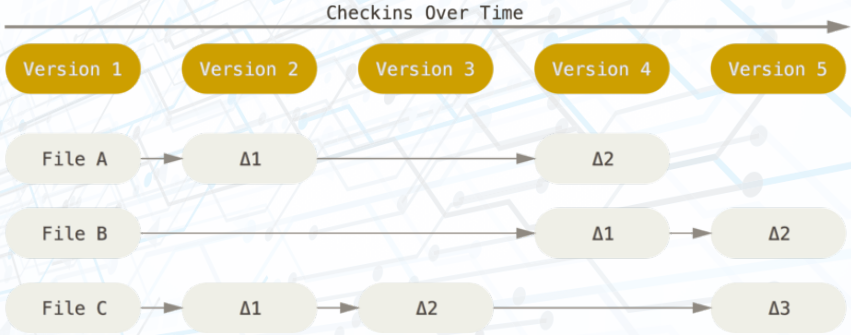
- Advantages
  - Everyone know what others are doing (at some degree)
  - Administrators have fine-grained control over the system
- Downside
  - Single point failure
- Having everything of your project in one place is not a good idea!

Distributed VCS

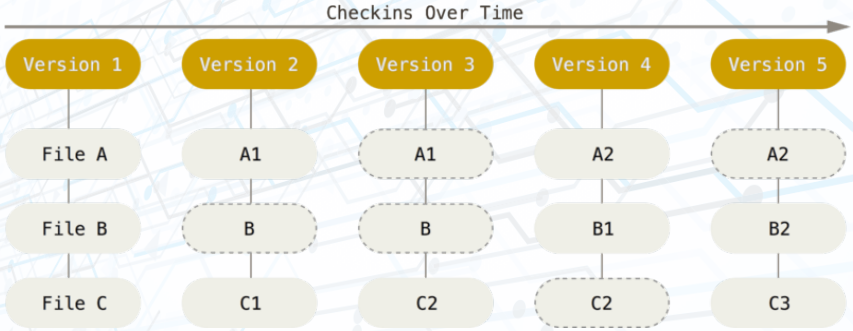
- DVCS: Git, Mercurial, Bazaar or Darcs
- Clients do not just checkout the snapshot of the project. They fully mirror the whole repository.



Delta-based VCS (CVS, Subversion, Perforce, Bazaar, and so on): store the difference overtime.



Git think about its data as a **stream of snapshots**.



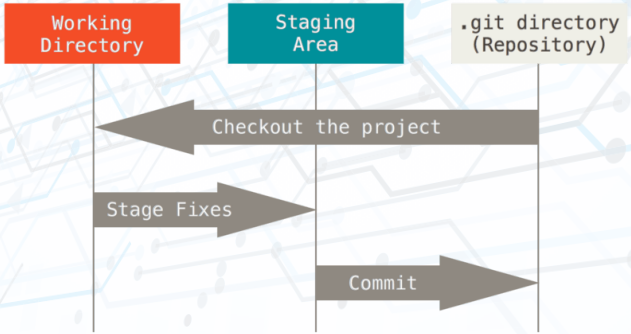
What is Git

- A **distributed** version control system
- Some use cases:
  - Keep a *history* of previous versions
  - *Develop simultaneously* on different branches
    Easily try out new features, integrate them into production or throw them out
  - **Collaborate** with other developers
    "push" and "pull" code from hosted repositories such as *Github*

What is Git

- A distributed version control system
  - Everyone can act as the "server"
  - Everyone mirrors the entire repository instead of simply checking out the latest version of the code (unlike svn)
- Many local operations
  - Cheap to create new branches, merge, etc.
  - Speed increases over non-distributed systems like svn

- Most operations in Git are local.
- **Integrity**: everything in Git is checksumed (SHA-1) before it is stored.
- Git only adds data
- Git workflow: three states



Basic Git workflow

- **Modify** files in your working directory.
- **Stage** files, adding snapshots of them to your staging area.
- **Commit**, which takes the files in the staging area and stores that snapshot permanently to your Git directory.

Getting started - Installing Git

- Linux: if you are on Ubuntu (Debian-based distribution)

```
$ sudo apt install git-all
```

- OSX: using XCode command line tools, using the Git installer, or brew (whatever fits you).
- Windows: using available installer `https://git-scm.com/download/win`
- Build from source.
  You will need to learn about the build system. It's fun.

Git commit checksum

- In Subversion each modification to the central repo increments the version # of the overall repo.
- In Git
  - In Git, each user commits changes to their local copy of the repo before pushing to the central server.
  - So Git generates a unique **SHA-1 hash** (40 character string of hex digits) for every commit.
  - Refers to commits by this **ID** rather than a version number.
  - Often we only see the first 7 characters:

```
1677b2d Edited first line of readme
258efa7 Added line to readme
0e52da7 Initial commit
```

Initial Git configuration

- Set the name and email for Git to use when you commit:

```
$ git config --global user.name "Bugs Bunny"
$ git config --global user.email bugs@gmail.com
```

Verifying

```
$ git config --list
```

- Set the editor that is used for writing commit messages

```
$ git config --global core.editor nano
```

it is vim by default (if you are using Linux)

- Mac/Linux line breaks

```
$ git config --global core.autocrlf input
```

Windows line breaks

```
$ git config --global core.autocrlf true
```

Creating a Git repository

- Create a new repository in the current directory

```
$ git init
```

This will create `.git` directory in your current directory.

```
$ git add filename
```

This will add the file into git database, so that you can manage and commit it.

```
$ git commit -m "commit message"
```

- Clone a repository

```
$ git clone repoURL localPath
```

Git commands

- clone: make a local copy of a repository
- add: add file contents to the staging area
- commit: record the snapshot of the staging area
- status: checking the status of files in the working directory and staging area
- diff: shows the difference of what is staged and what is modified but unstaged
- help: get info about particular command
- pull: fetch from the remote repository and merge to the current branch
- push: push new branches and data to the remote repository
- Other commands: init, reset, branch, checkout, merge, log, tag

Add and commit file

- The first time we ask a file to be tracked, and *every time before we commit a file*, we must add it to the staging area

```
$ git add Hello.cpp myLib.h
```

Takes a snapshot of these files, adds them to the staging area.
In older VCS, "add" means "start tracking this file". In Git, "add" means "add to staging area" so it will be part of the next commit.

- To move staged changes into the repo, we commit

```
$ git commit -m "Fixing bug 98"
```

- To undo changes on a file before you have committed it

```
$ git reset HEAD -- filename
$ git checkout -- filename
```

All these commands are acting on your local version of repo.

Viewing/undoing changes

- To view status of files in working directory and staging area

```
$ git status
```

- To see what is modified but unstaged

```
$ git diff
```

- To see a list of staged changes

```
$ git diff --cached
```

- To see a log of all changes in your local repo

```
$ git log
1677b2d Edited first line of readme
258efa7 Added line to readme
0e52da7 Initial commit
```

To show only the 5 most recent updates

```
$ git log -5
```

Removing a file

- To remove a file from the working tree and in the next commit

```
$ git rm filename
```

- To remove it from the next commit, but keep the file in the working tree

```
$ git rm --cached filename
```

Example

Let try a simple one.

.gitignore

- Specifies files that you don't want Git to track under version control
- Commonly used for compiled files, binaries, large asset files (e.g. images)
- Can use wildcards (e.g. *.pyc, *.png, Images/*, etc.)
- Be careful: if you add a file to .gitignore after it's already been tracked, potential issues
- A list of recommended .gitignore files:
  `https://github.com/github/gitignore`

Branching, merging, and rebasing

Branching

- What is a branch?
- Branching commands
- The HEAD pointer
- Basics of merging
- Basics of rebasing

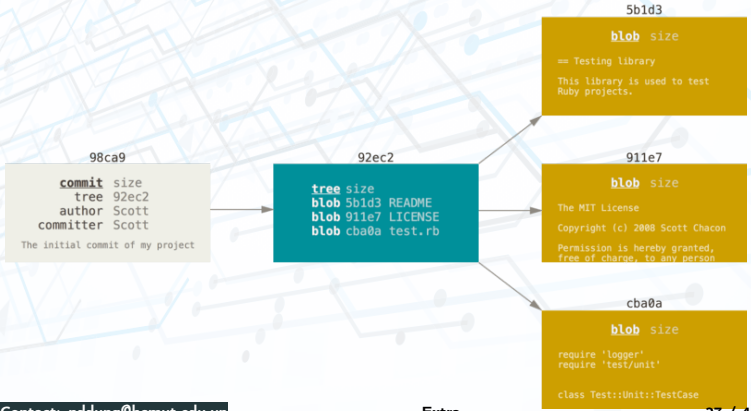
Branch

- Visualize a project's development as a "linked list" of commits.
- When a development track splits, a new branch is created.
- In Git, branches are actually just a pointer to these commits

Branch concept

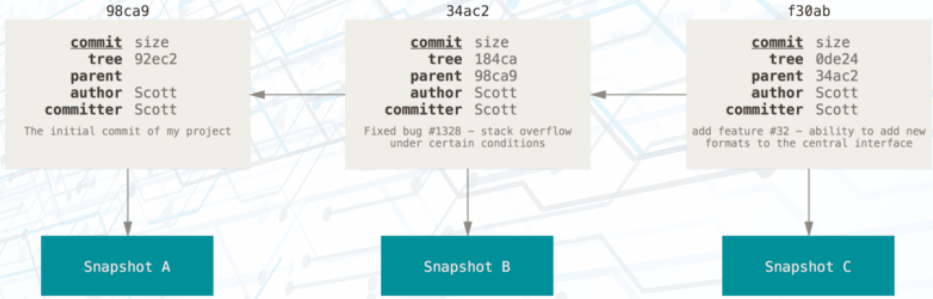
- Recall: Git stores its data as a series of **snapshots**

```
$ git add README test.rb LICENSE
$ git commit -m 'Initial commit'
```



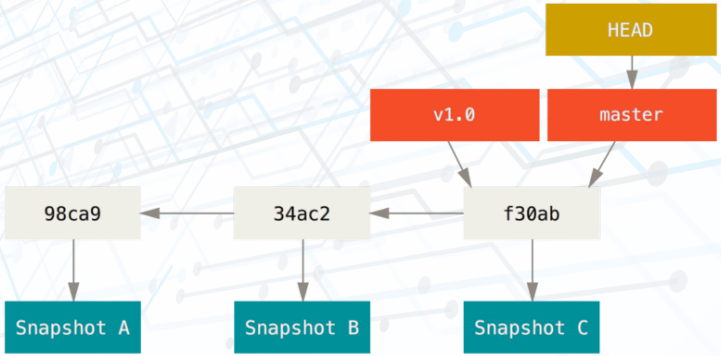
Branch concept

- Objects in Git: **blobs** (the contents of files), **tree** (content of the directory), and **commit** (pointer to the root of the tree).



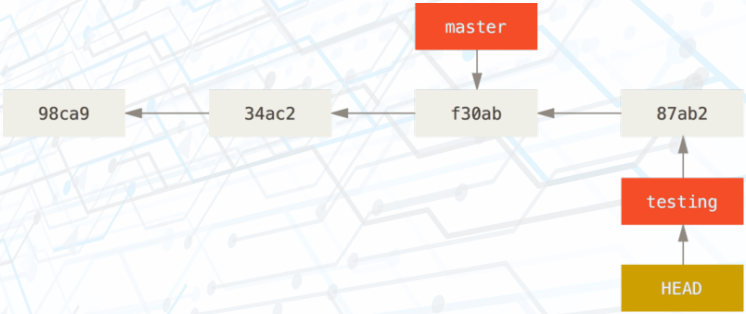
Branch concept

- A branch in Git is simply a lightweight movable pointer to one of these commits.
- The default branch name in Git is master.
- Git maintain a special pointer named **HEAD** which points to the current branch.



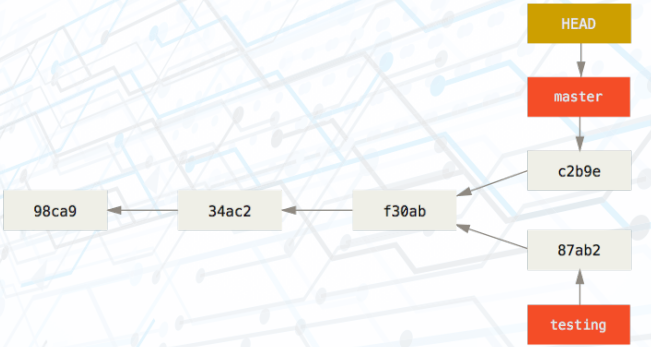
Branch concept

- When you commit, only the current branch is affected.



Branch concept

- Your project looks like a tree.



Branch commands

- Git uses branching heavily to switch between multiple tasks
  - To create a new local branch

```
$ git branch name
```

  - To list all local branches: (* = current branch)

```
$ git branch
```

  - To switch to a given local branch

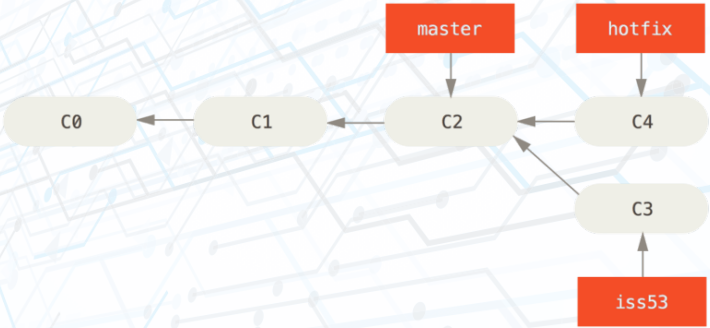
```
$ git checkout branchName
```

  - To merge changes from a branch into the local master

```
$ git checkout master
$ git merge branchName
```

Merging

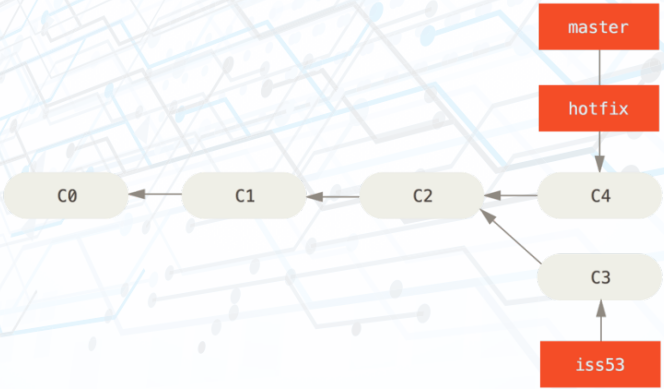
- Merging can become much more complicated than you think



Merging

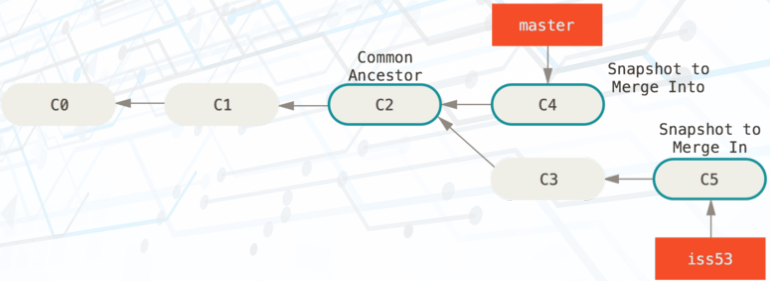
- Merge master and hotfix

```
$ git checkout master
$ git merge hotfix
```



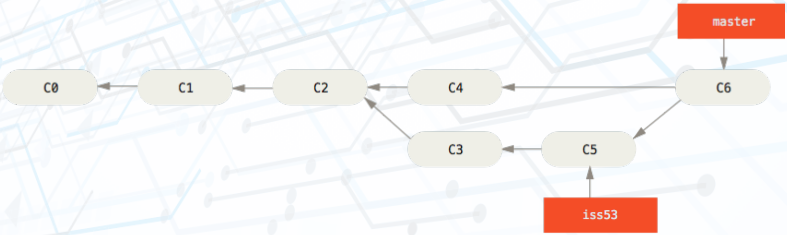
- Merge master and iss53

```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
```



Merging

- Git creates a new snapshot that results from this three-way merge
- Creates a new commit that points to it.
- This is referred to as a merge commit, and is special in that it has **more than one parent**.



Merge conflict

- Merge does not always process smoothly. This happen when you modify the same file on two different branches.
- We may see something like this

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Merge conflict

- Check for details

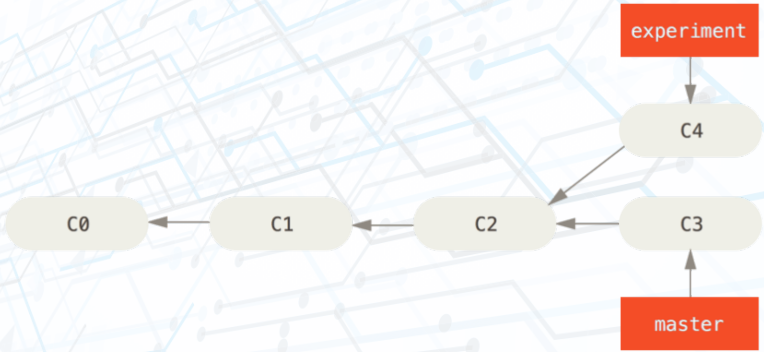
```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:      index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

- Remember: **mergetool** is your friend

```
$ git mergetool
```

Rebasing

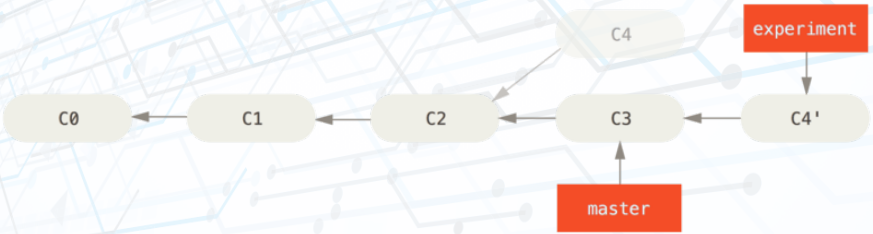
- Consider the following situation



Rebasing

- Instead of merging, we can perform rebase operation

```
$ git checkout experiment
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: added staged command
```



Working with remote

Working with remote

- Showing your remote

```
$ git clone https://github.com/schacon/ticgit
Cloning into 'ticgit'...
remote: Reusing existing pack: 1857, done.
remote: Total 1857 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (1857/1857), 374.35 KiB | 268.00 KiB/s, done.
Resolving deltas: 100% (772/772), done.
Checking connectivity... done.
$ cd ticgit
$ git remote
origin
```

- More information on your remote

```
$ git remote -v
origin  https://github.com/schacon/ticgit (fetch)
origin  https://github.com/schacon/ticgit (push)
```

Working with remote

- Add more remotes

```
$ git remote add pb https://github.com/paulboone/ticgit
$ git remote -v
origin  https://github.com/schacon/ticgit (fetch)
origin  https://github.com/schacon/ticgit (push)
pb      https://github.com/paulboone/ticgit (fetch)
pb      https://github.com/paulboone/ticgit (push)
```

- Fetching from the remote

```
$ git fetch remoteName
```

- Pushing to your remote

```
$ git push remoteName branchName
```

Inspecting a remote

- More information

```
$ git remote show origin
* remote origin
  Fetch URL: https://github.com/schacon/ticgit
  Push  URL: https://github.com/schacon/ticgit
  HEAD branch: master
  Remote branches:
    master                               tracked
    dev-branch                           tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

- Renaming and removing a remote

```
$ git remote rename pb paul
$ git remote remove paul
```

References

- The content is based on Git ebook.
- Images are taken from: `http://git-scm.com/book/en/`
- CSE 403.
- Charles Liu's material.