UECS2363 SOFTWARE CONSTRUCTION AND CONFIGURATION
CHAPTER 4 : Git

LOO YIM LING
ylloo@utar.edu.my

UTAR

Version control systems

- **Version control** (or **revision control**, or **source control**) is all about managing multiple versions of documents, programs, web sites, etc.
  - Almost all "real" projects use some kind of version control
  - Essential for team projects, but also very useful for individual projects

Version control systems

- Some well-known version control systems are CVS, Subversion, Mercurial, and Git

  - CVS and Subversion use a "central" repository; users "check out" files, work on them, and "check them in"

  - Mercurial and Git treat all repositories as equal

- Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion

Why version control?

- For working by yourself:
  - Gives you a "time machine" for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project

UTAR

Why version control?

- **For working with others:**

  - Greatly simplifies concurrent work, merging changes

- **For getting an internship or job:**

  - Any company with a clue uses some kind of version control

  - Companies without a clue are bad places to work

Why Git?

- Git has many advantages over earlier systems such as CVS and Subversion

  - More efficient, better workflow, etc.

  - See the literature for an extensive list of reasons

  - Of course, there are always those who disagree

- Best competitor: Mercurial

  - Same concepts, slightly simpler to use

  - Much less popular than Git

Download and install Git

- **http://git-scm.com/downloads**

- **Here's one from StackExchange: http://stackoverflow.com/questions/315911/git-forbeginners-the-definitive-practical-guide#323764**

- **Note: Git is primarily a command-line tool, but GUI tool is available too.**

Introduce yourself to Git

- Enter these lines (with appropriate changes):

  - ```
    git config --global user.name "John Smith"
    ```

  - ```
    git config --global user.email jsmith@seas.upenn.edu
    ```

- You only need to do this once

UTAR

Introduce yourself to Git

- If you want to use a different name/email address for a particular project, you can

- change it for just that project

  - `cd` to the project directory

  - Use the above commands, but leave out the `--global`

Create and fill a repository

1) `cd` to the project directory you want to use

2) Type in `git init`

- This creates the repository (a directory named `.git`)
- You seldom (if ever) need to look inside this directory

Create and fill a repository

3)Type in `git add .`

- The period "**.**" at the end is part of this command!
    - Period "**.**" means "this directory"
- This adds all your current files to the repository

4)Type in `git commit -m "Initial commit"`

- You can use a different commit message, if you like

UTAR

Clone a repository from elsewhere

- `git clone URL`
- `git clone URL mypath`
  - These make an exact copy of the repository at the given URL
- `git clone`
- `git://github.com/rest_of_path/file.git`
  - Github is the most popular (free) public repository

Clone a repository from elsewhere

- **All repositories are equal**

  - But you can treat some particular repository (such as one on Github) as the "master" directory

- **Typically, each team member works in his/her own repository, and "merges" with other repositories as appropriate**

The repository

- Your top-level **working directory** contains everything about your project

  - The working directory probably contains many subdirectories— source code, binaries, documentation, data files, etc.

  - One of these subdirectories, named `.git`, is your repository

- At any time, you can take a "snapshot" of everything (or selected things) in your project directory, and put it in your repository

The repository

- This "snapshot" is called a **commit object**

- The commit object contains (1) a set of files, (2) references to the "parents" of the commit object, and (3) a unique "SHA1" name

- Commit objects do not require huge amounts of memory

- You can work as much as you like in your working directory, but the repository isn't updated until you `commit` something

`init` and the `.git` repository

- When you said `git init` in your project directory, or when you cloned an existing project, you created a repository

  - The repository is a subdirectory named `.git` containing various files

  - The dot indicates a "hidden" directory

  - You do *not* work directly with the contents of that directory; various git commands do that for you

  - You *do* need a basic understanding of what is in the repository

Making commits

- You do your work in your project directory, as usual

- If you create new files and/or folders, they are not tracked by Git unless you ask it to do so

  - `git add newFile1 newFolder1 newFolder2 newFile2`

- Committing makes a "snapshot" of everything being tracked into your repository

  - A message telling what you have done is required

Making commits

- `git commit -m "Uncrevulated the conundrum bar"`

- `git commit`

- This version opens an editor for you the enter the message

- To finish, save and quit the editor

- Format of the commit message

  - One line containing the complete summary

  - If more than one line, the second line must be blank

Commits and graphs

- A **commit** is when you tell git that a change (or addition) you have made is ready to be included in the project

- When you commit your change to git, it creates a **commit object**

  - A commit object represents the complete state of the project, including all the files in the project

  - The *very first* commit object has no "parents"

Commits and graphs

- Usually, you take some commit object, make some changes, and create a new commit object; the original commit object is the parent of the new commit object
  - Hence, most commit objects have a single parent
- You can also **merge** two commit objects to form a new one
  - The new commit object has two parents
- Hence, commit objects form a **directed graph**
- Git is all about using and manipulating this graph

Working with your own repository

- A **head** is a reference to a commit object

- The "current head" is called **HEAD** (all caps)

- Usually, you will take **HEAD** (the current commit object), make some changes to it, and commit the changes, creating a new current commit object

  - This results in a linear graph: A →B →C →…→HEAD

Working with your own repository

- You can also take any previous commit object, make changes to it, and commit those changes

  - This creates a <u>branch</u> in the graph of commit objects

- You can <u>merge</u> any previous commit objects

  - This joins branches in the commit graph

Commit messages

- In git, "Commits are cheap." Do them often.

- When you commit, you must provide a oneline message stating what you have done

  - Terrible message: "Fixed a bunch of things"

  - Better message: "Corrected the calculation of median scores"

- Commit messages can be very helpful, to yourself as well as to your team members

- You can't say much in one line, so commit often

UTAR

Choose an editor

- When you "commit," git will require you to type in a commit message

- For longer commit messages, you will use an editor

- The default editor is probably **vim**

- To change the default editor:

  - ```
    git config --global core.editor /path/
    to/editor
    ```

- You may also want to turn on colors:

  - ```
    git config --global color.ui auto
    ```

Working with others

- All repositories are equal, but it is convenient to have one central repository in the cloud

- Here's what you normally do:

  - Download the current HEAD from the central repository

  - Make your changes

  - Commit your changes to your local repository

  - Check to make sure someone else on your team hasn't updated the central repository since you got it

  - Upload your changes to the central repository

Working with others

- If the central repository has changed since you got it:

  - It is your responsibility to **merge your two versions**

    - This is a strong incentive to commit and upload often!

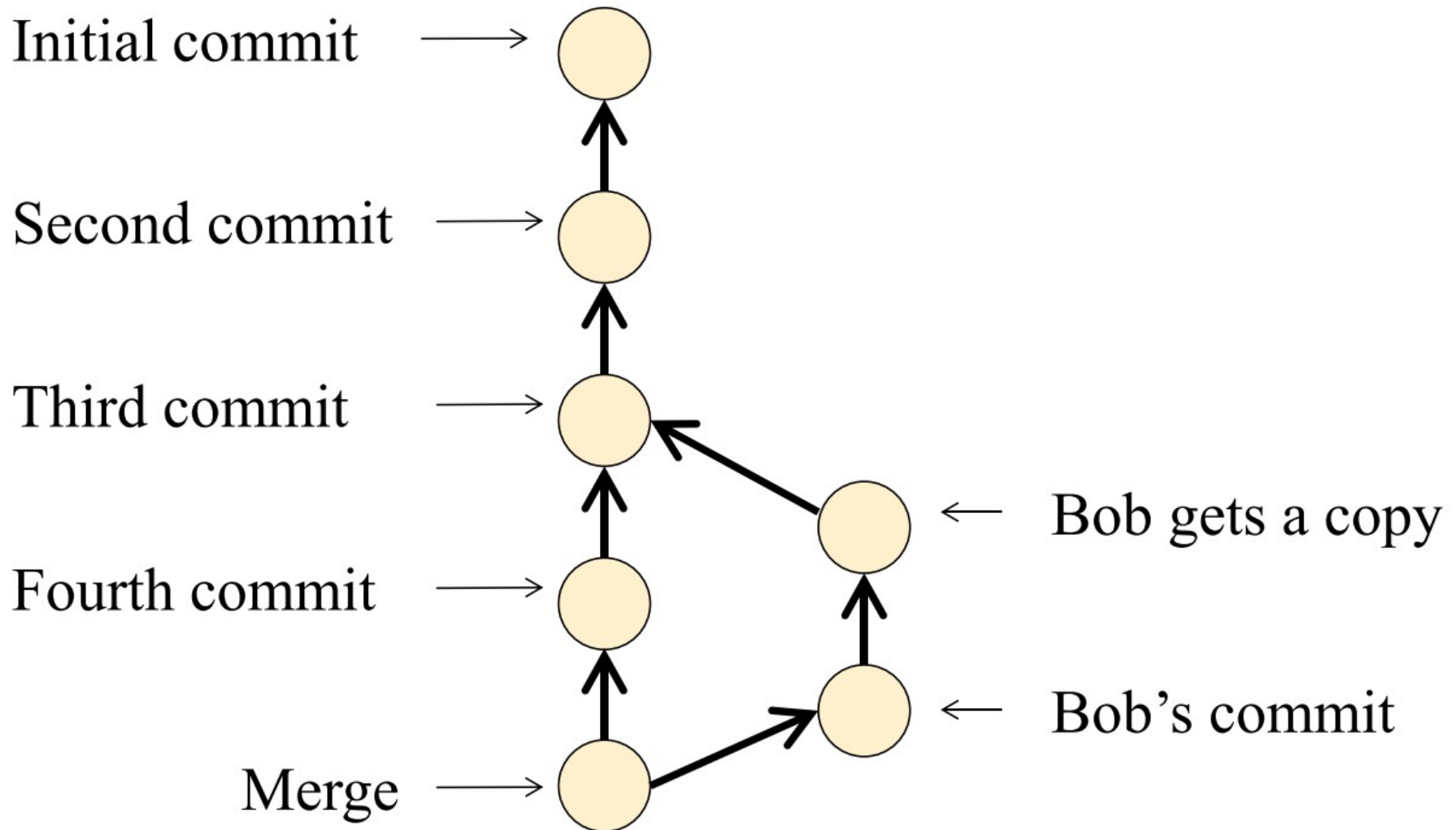 - Git can often do this for you, if there aren't incompatible changes

Typical workflow

- `git pull remote_repository`
  - Get changes from a remote repository and merge them into your own repository

- `git status`
  - See what Git thinks is going on
  - Use this frequently!

Typical workflow

- **Work on your files (remember to add any new ones)**

- `git commit –m "What I did"`

- `git push`

Multiple versions

Initial commit →

Second commit →

Third commit →

Fourth commit →

← Bob gets a copy

← Bob's commit

Merge →

UTAR

Keeping it simple

- If you:
  - Make sure you are current with the central repository
  - Make some improvements to your code
  - Update the central repository before anyone else does
- Then you don't have to worry about resolving conflicts or working with multiple branches
  - All the complexity in git comes from dealing with these

Keeping it simple

- **Therefore:**

  - **Make sure you are up-to-date before starting to work**

  - **Commit and update the central repository frequently**

- **If you need help: https://help.github.com/**

END OF LECTURE 05