Preliminary Project

1 Goal

After finishing this project, you can:

1. Get familiar with Linux
2. Get familiar with GNU C Compiler (GCC)
3. Get familiar with Git
4. **Give a detailed report for the project**

2 Prerequisites

2.1 Operating System

You need a **Linux** Operating System to run *gcc* and *git*. The recommended operating system is `Ubuntu 16.04 LTS`. If you do not know how to install, you can find the tutorial at the appendixes.

> It is only **optional** for you to install a new operating system. If you *already* have an accessible Linux, you can **SKIP** this.

Other Unix-like systems (e.g. Mac OS) should also work but they are not recommended, although it is **still OK** if you insist on it and work well with it. If you don't have such an operating system, a new installation is required.

For new users, a desktop helps a lot when trying a new OS. Remember to **install GNOME desktop** (the installer will do it by default) if you are not familiar with the new OS.

2.2 GCC Version

The **REQUIRED** GCC version is `5.4.0`. You need to make sure that the version is correct. Try on your own risk if you still want another version.

2.3 Git version

Git version is not compulsory. Any version is OK if you can perform basic operations (i.e. *clone, pull, push, commit,* etc.)

3 Contents

3.1 Installation

First install Ubuntu on your computer or on a virtual machine instance in VirtualBox or VMware workstation. Use Google to help you if you do not know how to this.

> If you choose to install on your physical computer for the first time, get someone to help you or follow a trustful tutorial **WHENEVER** you do not know how to make a choice. An arbitary choice may destroy anything important on your computer.

Then you should install Git.

> If you completely know nothing of operating here, directly go on to next section 3.2. This tutorial will help you after you are familiar with some basic operations and commands.

See appendix A.2 to get help of installing git. For quick installation, just type `sudo apt-get install git` and continue.

3.2 Ubuntu basic operations

Here are some basic operations for Ubuntu.

> This part is **ONLY** for new users. Skip it if you know everything.

3.2.1 Open GNOME terminal

The default desktop of Ubuntu is GNOME. The shortcut for opening a terminal is `Ctrl+Alt+T`.

3.2.2 Basic commands in terminal

- Try `pwd` command to show which directory path you are in.
- Try `ls` command to show everything in your current directory.
- Try `mkdir` command to create a new directory.
- Try `cd` command to change path to another directory.
- Try `cat` command to show the content of a pritable file (i.e. text file).

3.2.3 Installing software with the Advanced Package Tools (APT)

When you need to install a software, a nice choice is to check whether it is in the APT repository. Most of the handy softwares are compiled and configured in advance into binary files. You just need to pull them down to your own computer.

Use USTC's software mirror

Once the system has been installed, the default repository is Ubuntu's main repository on `ubuntu.com`. This is OK, but it is much slow when we try to pull down softwares. The best choice here is to use USTC's mirror.

On the official website, there's a tutorial. Another way is:

1. Click on the 'Settings' button on the right-top corner of your desktop.
2. Click on 'Softwares & Updates' in the 'System' Section.
3. In the pop-up dialogue frame, click on the dropdown list beside 'Download from' label. Choose 'Other', then find 'mirrors.ustc.edu.cn' in 'China'.

Install *cmake* onto your Ubuntu

Now try to install `cmake`:

1. First directly try `cmake` command in the terminal. If not on the computer, `cmake` command would fail, and together with which a message shows up: `The program 'cmake' is currently not installed...`
2. Now try to run this command: `sudo apt-get install cmake`. Type in `yes` or `Y` when asked whether to continue.
3. Wait for the command ends.
4. Try `cmake` now. The terminal now shows the usage of `cmake`.

Now, if you did not install `git` in 3.1 section, try to install it by yourself!

3.2.3 SSH key generation

You need SSH key for Git connection. Now try to generate a new pair of keys:

```
(shell) $ ssh-keygen -t rsa
```

Then press [Enter] and wait.

Now your key is at `~/.ssh/id_rsa` (private) and `~/.ssh/id_rsa.pub` (public). Once your public key is added to the Git server, you will have priviledge to push/pull your repositories.

> If you want to change the default output file, use '-f' option. See manual pages of `ssh-keygen` for details.

3.2.4 GNOME Editor

The command for opening the default editor is `gedit`. Now try to do the following:

1. Create a testing directory and change directory into it.

   ```
   (shell) $ mkdir test
   (shell) $ cd test
   ```

2. Use GNOME editor to create a file with name `test.c`:

   ```
   (shell) $ gedit test.c
   ```

Now an editor frame should show up.

3. Write a piece of C source code into this file. For example:

```
#include <stdio.h>
int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

Click 'Save' and close the window.

3.3 Compilation with GCC

Now you can compile the testing C source code. The entire compilation process can be done with one single command, but now we try to break down into four stages.

3.3.1 Pre-processing

Try the following command to get the result of pre-processing the source code.

```
(shell) $ gcc -E test.c -o test.i
```

Now file `test.i` contains the extraction of macro definitions and header files.

3.3.2 Compliation

This stage is when GCC really compiles the code. Try:

```
(shell) $ gcc -S test.i -o test.s
```

Now the test.s contains the generated assembly instructions.

3.3.3 Assembly

Now this stage the compiler uses an assembler to translate the assembly instructions into object code, which consists of the actual instructions that can be run by the target processor. Try:

```
(shell) $ gcc -c test.s -o test.o
```

3.3.4 Linking

Now the instructions are incomplete, since part of the program is missing (the `printf` function). The implementation is actually in `libc.so.6` file under path `/usr/lib`. The compiler links it here.

```
(shell) $ gcc test.o -o test
```

Eventually, an executable binary file is here.

3.3.5 Running the output

Use `./test` to run. Here `.` means the current directory, which tells Ubuntu where to find the executable file.

> If you want to start an executable by only calling its name (i.e. use `test` instead of `./test`), then the directory of the executable file MUST be in the **$PATH** variable.

3.4 Working with Git

Now after section 3.2.3 you should have Git on your OS. Now we try to use it.

3.4.1 Git basics

Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. (from Wikipedia)

A git repository is where git manages a project of files. In most cases, a repository means a directory. This directory differs from other directories in that it keeps a hidden subdirectory '.git/', which contains the metadata and data of this repository.

We do not need to manually do operations in this hidden directory. `git` command helps us with anything.

3.4.2 Create a new local directory

Now change directory to where you compiled the 'helloworld' program in section 3.3. Try to create a new git repository with command:

```
(shell) $ git init
```

3.4.3 Keep track of files

Now the repository you've just created keeps track of nothing. That is, git do not know anything in this repository directory. Try to keep track of a file with the command:

```
(shell) $ git add test.c
```

It is supported to add a whole directory or subdirectory to git repository as well. Add everything in the current directory with:

```
(shell) $ git add .
```

> Here '.' means the current directory.

3.4.4 Commit changes

Now git keeps track of everything here, and you need to commit. Commit is more like a checkpoint or a snapshot when the project evolves. Once a commit is done, the repository is able to roll back to this status in the future. What is more, git enables you to show the differences of two commits, which helps developers know which part is exactly changed between versions.

Try to commit with:

```
(shell) $ git commit -m "First commit"
```

Each time you commit, a note / message is **required** to comment on this commit, which helps to determine the purpose of each commit in the future.

The '-m' option helps to include the note in the command. Committing without the option will trigger the opening of a command-line editor (*e.g. nano, vi*) to force you to write the note.

> You can add all changes and commit in one command with option '-a':
>
> ```
> (shell) $ git commit -am "First commit"
> ```
>
> With this, git automatically executes `git add .` and commit.

3.4.5 Branching

So, what is branching? It is hard to define in a few words. Totally, it means a divergent serial of work, in the form of a tree. A more detailed introduction is Here on the official website.

When the repository is created, the default branch you are on is *master*. Now create a new branch forked from *master* called *adding*.

```
(shell) $ git branch adding
```

Now shift to branch *adding*.

```
(shell) $ git checkout adding
```

> Note: These two commands can be shortened as:
>
> ```
> (shell) $ git checkout -b adding
> ```

Now make changes to file `test.c` as follows:

```
#include <stdio.h>

int add(int x, int y)
{
    return x + y;
}

int main()
{
    printf("Hello, world!\n");
    printf("1 + 2 = %d\n", add(1, 2));
    return 0;
}
```

3.4.6 Merging

Now part of your work (the `add` function) is finished. Commit them, and merge the changes to the main branch *master*.

```
(shell) $ git commit -am "add func complete"
(shell) $ git checkout master
(shell) $ git merge adding
```

Then *master* branch merges all changes in *adding*. Branch *adding* now is useless. Delete it with command:

```
(shell) $ git branch -d adding
```

3.4.7 Git log and rolling back

Following the tutorial means now the source code has been changed. Now we can look back to history. Try:

```
(shell) $ git log (--oneline)
```

Here with '--oneline' option does not show the details. For each commit, there's a hashing ID. You can go back to any commit without doing anything to any commits behind the target commit.

```
(shell) $ git checkout (commitID)
```

After this, you can fork another branch and do something else.

If recent changes are determined to be destroyed, use:

```
(shell) $ git reset --hard/--soft (oldCommitID)
```

This `reset` command will **DESTROY ALL** newer commits. Think before you act.

The difference for option '--hard' and '--soft' is whether git will remove all newer changes in the files. However, commits will all be destroyed.

> If only one commit is considered to cancel, try `git revert`. Search online for more details.

3.4.8 Remote operations

Now finally we come to the remote repositories. What we mentioned above only works in your own **LOCAL** repository. How do multiple people cooperate?

Git servers hold shared repositories for multiple people to use. Users have the operation of uploading (`push`) and downloading (`pull`) latest changes.

Cloning remote repositories

It is common that you do not start from scratch. In most cases, when a repository is created on the server, you just need to clone to local.

```
(shell) $ git clone git@domain:/path/to/repo
```

For example, you can clone the *tensorflow* source code from Github:

```
(shell) $ git clone git@github.com:tensorflow/tensorflow.git
```

As for your own repository URL, find it on the Git server.

Push & Pull

Now assume that you have a remote repository on a Git server, you've cloned the repository to local and made changes and committed. Just push your changes to the server.

```
(shell) $ git push origin master
```

Here, `origin` is an alias for your remote URL. This is automatically set when you cloned the remote repository. The last parameter here means which branch you are working with.

Replacing `push` with `pull` will do the opposite. If remote repositories contain newer changes, you can pull them down.

4 Assignments

In this preliminary project, you should complete:

1. You have an available Linux OS.
2. Generate your own SSH key pairs.
3. Create an account whose name is your student ID (e.g. PB16011XXX) on the Git server, add your public key onto it.
4. Create a new repository for this project, named "pre-project".
5. Clone the remote repository to your desktop with `git`.
6. Add the 'Hello, world' program mentioned above as the first commit.
7. Follow the 4 stages of compilation for 4 different commits.
8. Push all commits to the server when you finished.
9. **Write a report for this project**, including problems in the project. For each problem, you should explain **what it is, where or how it happens and how you solved it**.

A Appendix

A.1 How to install Ubuntu 16.04 LTS

Official tutorial is HERE.

Chinese tutorial can be found easily, e.g. HERE

> Tips:
>
> 1. You do not need to download ISO file from Ubuntu official site. Use USTC's open-source software mirror instead: http://mirrors.ustc.edu.cn/
> 2. Whenever you do not know how to make a choice, ask someone to help you. Be careful to choose your installation partition on your disk.

A.2 How to install GCC 5.4.0 and Git from repository or source code

GCC is pre-installed in various Linux releases. In Ubuntu 16.04, the default GCC version is 5.4.0. As to Git, you need to install by yourself with the commmand `sudo apt-get install git`.

The official tutorial for GCC and Git can be easily found at official websites. It is strongly recommended to download from

Building from source code may be troubling to some extent. You should first get the tarball of your desired version. The mirror that is officially recommended for Git is https://mirrors.edge.kernel.org/pub/software/scm/git/. Follow the steps below:

1. First, use `wget` to download the desired tarball. For example if the 2.9.5 version of Git is desired, use:
   `(shell) $ wget https://mirrors.edge.kernel.org/pub/software/scm/git/git-2.9.5.tar.gz`

2. Configure your environment. Use `./configure` to do this. If anything is required but not found, it teaches you to install.

   > Remember to use option `--prefix=` if you do not want to install to the default location `/usr/local`

3. `make` and `make install`

   > It is often the case that you need to `sudo` when installing to system paths, i.e. `sudo make install` works if `make install` doesn't.