Computer Setup Programming

Robin Hellmers

November 7, 2021

Contents

1 Git

1.1 Multiple repositories

1.1.1 Repository across 2 computers

Relevant StackOverflow **answer**

One might want to develop code on two machines, without using a server such as Github or similar.

Lets say you have created a repository and developed on **Computer 1**. That could be only locally or still through some server which you might not want to access through the other computer.

So you have a repository and the working directory on **Computer 1**, could also be on **Computer 2**, does not matter:

`~/<pathToRepo>/<repoName>`

which contains a `.git` with all the git related revision history.

Now you want to be able to push to **Computer 2**. In order to do that you need

Computer 1:

- Normal repository (non-bare) with the working directory

Computer 2:

- Normal repository (non-bare) with the working directory.
- Bare repository

The usage will be:

1. From **Computer 1** working directory: Push to the bare repository on **Computer 2**
2. From **Computer 2** working directory: Pull from bare repository on **Computer 2**.

or the other way around

1. From **Computer 2** working directory: Push to the bare repository on **Computer 2**.
2. From **Computer 1** working directory: Pull from the bare repository on **Computer 2**

Instructions

Lets say you have the original repository with its working directory on **Computer 1** with its `.git` directory in `~/<pathToRepo>/<repoName>/`

1. **Computer 1**: Make a bare clone of the repository with the name extension `.git`

   ```
   git clone --bare ~/<pathToRepo>/<repoName> ~/<pathToRepo>/<repoName>.git
   ```

2. **Computer 1**: Copy it over to **Computer 2**, using SSH.

   ```
   scp -r ~/<pathToRepo>/<repoName>.git <username>@<ipAddress>:<relativePathBare>
   ```

   or

```
scp -r ~/<pathToRepo>/<repoName>.git ssh://<username>@<ipAddress>/<fullPathBare>
```

3. `Computer 2`: Clone the bare repository into a non-bare repository.

   ```
   git clone ~/<relativePathBare>/<repoName>.git ~/<pathToStoreRepo>/.
   ```

1.2 Adding a remote

1.2.1 SSH

Create SSH keys

Create SSH keys by entering a passphrase connected to the SSH keys, after running:

```
ssh-keygen -f ~/.ssh/id_ecdsa -t ecdsa -b 521
```

In Ubuntu, run the `ssh-agent` with the command:

```
eval "$(ssh-agent -s)"
```

Add the SSH keys to the `ssh-agent` by entering the passphrase after:

```
ssh-add ~/.ssh/id_ecdsa
```

Github

Copy the public key, that is all of the content in `id_ecdsa.pub`. If in WSL Ubuntu, copy the content with:

```
clip.exe < ~/.ssh/id_ecdsa.pub
```

Go to Github, then:

`Settings` → `SSH and GPG keys` → `Add copied public key`

Now you can clone a repo:

```
git clone git@github.com:<username>/<reponame>.git
```

Say yes to the fingerprint.

1.3 Configurations

1.3.1 Log - Commit history

In order to have a good git history tree visualization in the terminal, use the `.gitconfig` from Github. This gives three different commands:

```
[alias]
    show-gitignore = git ls-files -ci --exclude-standard
    apply-gitignore = !git ls-files -ci --exclude-standard -z | \
                      xargs -0 git rm --cached
    apply-gitignore-remove = !git ls-files -ci --exclude-standard -z | \
                             xargs -0 git rm --cached
    lg = log --all --graph --abbrev-commit --decorate \
      --format=format:'%C(bold blue)%h%C(reset) - %C(bold green)(%ar)%C(reset)'\
      ' %C(white)%s%C(reset) %C(dim white)- %an%C(reset)%C(auto)%d%C(reset)'
    lg2 = log --all --graph --abbrev-commit --decorate \
      --format=format:'%C(bold blue)%h%C(reset) - %C(bold cyan)%aD%C(reset)'\
      ' %C(bold green)(%ar)%C(reset)%C(auto)%d%C(reset)%n''          '\
      '%C(white)%s%C(reset) %C(dim white)- %an%C(reset)'
    lg3 = log --all --graph --abbrev-commit --decorate \
      --format=format:'%C(bold blue)%h%C(reset) - %C(bold cyan)%aD%C(reset)'\
      ' %C(bold green)(%ar)%C(reset) %C(bold cyan)(committed: %cD)%C(reset)'\
      ' %C(auto)%d%C(reset)%n''          %C(white)%s%C(reset)%n''          '\
      '%C(dim white)- %an <%ae> %C(reset) %C(dim white)(committer:'\
      ' %cn <%ce>)%C(reset)'
```

1.3.2 Apply gitignore

NOTE: *This will remove the files from remote repo and thereby remove the files from other computers.*

Show the files which currently apply to the `.gitignore`:

```
git ls-files -ci --exclude-standard
```

Have a command which

1. Looks up the `.gitignore` files
2. Removes them locally
3. Stages the changes (add): the removed files

Add this line to `.gitconfig` under `[alias]` in order to have a short command:

```
apply-gitignore-remove = !git ls-files -ci --exclude-standard -z | xargs -0 git rm --cached
```

Called with:

```
git apply-gitignore-remove
```

Then commit the removed files.

1.3.3 Multiple git users

One might want to have multiple users, e.g. one for work with the work email and a private with private email.

As standard, when doing the commands:

```
git config --global user.name
```

```
git config --global user.email
```

It created the default user to the `~/.gitconfig` by appending

```
[user]
    name = <Your name>
    email = <Your email>
```

You can create another local configuration file `~/<pathToNewGitDirectory>/.gitconfig`, with the other user:

```
[user]
    name = <Other name>
    email = <Other email>
```

Open up `~/.gitconfig` and add `[includeIf ...]` with your new git configuration file path replaced:

```
[includeIf "gitdir:~/<pathToNewGitDirectory>/"]
    path = ~/<pathToNewGitDirectory>/.gitconfig
```

If you are in that directory or any sub-directory specified, your user will be `Other name` with `Other email`. Verify this by going to that specified path and type:

```
git config user.name
git config user.email
```

2 Visual Studio Code

Download and install Visual Studio Code on Windows

Go to **Extensions** with `Ctrl + Shift + X`.

Search for `Remote - WSL` and press **Install**.

You can now open any file or directory within Ubuntu by using `code <file/directory>`.

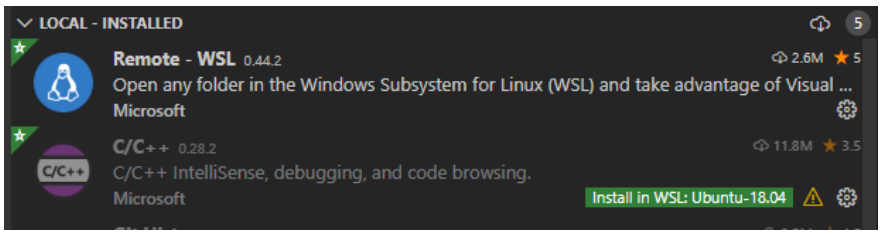
See Visual Studio Code's connection to the Ubuntu WSL by clicking on the **Remote Explorer Icon** to the left. Make sure that the dropdown menu at the top shows **WSL Targets**.

If there is a green symbol at your Ubuntu distribution, it is connected. If not, right click and press **Connect to WSL**.

You can also access your Ubuntu distribution terminal by clicking `Terminal \ra New Terminal`. Press the **dropdown arrow button** at the top right of the newly opened terminal and press **Ubuntu**.

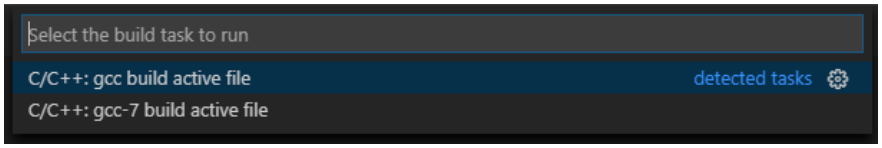
2.1 C programming

1. Install the necessary tools by running
   - `sudo apt install build-essential` for `gcc`
   - `sudo apt install gdb` for `gdb`
2. Create a new folder in the WSL where you create a C file `helloworld.c`. New folder is necessary for Visual Studio Code to realise that there is a C compiler to setup later on as it uses the open file to do the configurations.
3. Open up Visual Studio Code
4. Install the `C/C++` extension from Microsoft.
5. Press on the new icon on the left, `Remote explorer`. Right-click the distribution (e.g. `Ubuntu-18.04`) and press `Connect to WSL`. A new window will appear with some connection to the WSL.
6. Press on the extension icon the left in the new window. Press `Install in WSL: <distribution>` button on the `C/C++` extension.



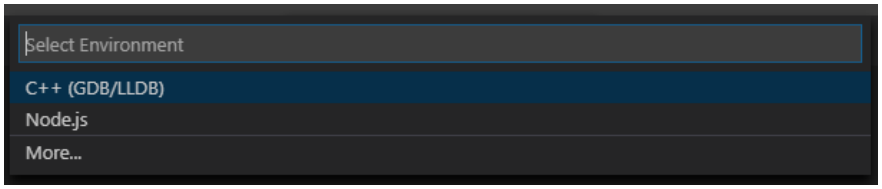
7. By now it might prompt that you have to reload the window. Press that button.
8. Open up the folder you created the main C file in. `File` → `Open Folder...`
9. Open up the `helloworld.c` file in the file explorer.
10. Press `Terminal` → `Configure Default Build Task...`. In the dropdown list that should appear, choose `C/C++: gcc build active file` (Not gcc-7). A file `tasks.json` will be created and opened up.

- No edits of the `tasks.json` is required for single file compilation with `gcc`.
- Edits are required for multi-file compilation with `gcc`.



2.1.1 Single file compilation

11. Do not edit the `tasks.json`
12. Build file with `Ctrl+Shift+b`. Press the `+` sign at the terminal to open a new terminal. Run the file `./helloworld` to test that everything is working.
13. Now onto debugging. Press `F5` or `Run` → `Start Debugging`. In the drop-down list that should appear, choose `C++ (GDB/LLDB)`. A file `launch.json` will be created and opened up.



14. Do not edit the `launch.json`.
15. Down at the `Output` and `Terminal`, press the three dots `...` and choose `Debug Console` in which one can run the standard `gdb` commands.

2.1.2 Multi-file compilation

Here is some info about setting up Visual Studio Code to build and debug projects including multiple files:

https://dev.to/talhabalaj/setup-visual-studio-code-for-multi-file-c-projects-1jpi

Here is an example of a `Makefile` I have used. Remember to use the `-g` flag if you want to debug. Also available here **https://github.com/robinhellmers/programming_setup**.

This `Makefile` is based on the following structure.

- `Makefile` in the main project folder.
- Four sub-folders: `bin`, `src`, `include`, `lib`
- Executable `.out` files in `bin`.
- Main `.c` files in `src`.
- Extra `.c` used as libraries in `lib`.
- All `.h` header files in `include`.

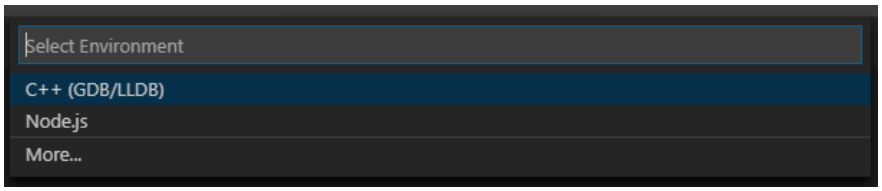
```
CC := gcc
CFLAGS := -pthread -g

BIN := bin
SRC := src
INCLUDE := include
LIB := lib

all: $(BIN)/server.out $(BIN)/client.out

$(BIN)/server.out: $(SRC)/server.c $(LIB)/*.c $(INCLUDE)/*.h
    $(CC) $(CFLAGS) -I$(INCLUDE) $^ -o $@

$(BIN)/client.out: $(SRC)/client.c $(LIB)/*.c $(INCLUDE)/*.h
    $(CC) $(CFLAGS) -I$(INCLUDE) $^ -o $@

clean:
    rm $(BIN)/server.out $(BIN)/client.out



# ${wildcard pattern}
# "wildcard" will list every file that follows the "pattern"
#
# Lets say we have the files hello.c hello.h goodbye.c goodbye.h
# ${wildcard *.c} will result in: hello.c goodbye.c
```

After creating one for the specific project, continue with the **Visual Studio Code** configuration:

11. The `tasks.json` must be edited according to the following.
    - Might have to check if there is some information in the generated `tasks.json` about the version number.
    - Code also available here: **https://github.com/robinhellmers/programming_setup** in the `.vscode` folder.
    - This edit will require a `Makefile` with an `make all` command for compiling all the different files together.
    - The label `"label": "build"` can be changed to any other, which will be used in the debugger config file `launch.json` later on. Same label will appear as a dropdown list later on.

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "shell",
            "label": "build",
            "command": "make all",
            "group": {
                "kind": "build",
                "isDefault": true
            },
            "problemMatcher": "$gcc"
        }
    ]
}
```

12. Build file with `Ctrl+Shift+b`. Press the `+` sign at the terminal to open a new terminal. Run the file `./helloworld` to test that everything is working.
13. Now onto debugging. Press `F5` or `Run` → `Start Debugging`. In the drop-down list that should appear, choose `C++ (GDB/LLDB)`. A file `launch.json` will be created and opened up.'



14. The `launch.json` must be edited according to the following.
    - Might have to check if there is some information in the generated `tasks.json` about the version number.
    - Code also available here: **https://github.com/robinhellmers/programming_setup** in the `.vscode` folder.
    - Set the prelaunch task `"preLaunchTask": "build"` to the label you set in the `tasks.json`, in this case to `"build"`. This will do the compilation according to our specification in the `tasks.json` and thereby compile with the `Makefile`.
    - Set which program to debug with `"program": "${workspaceFolder}/bin/${fileBasenameNoExtension}.out"`.
      This must be adjusted according to the `Makefile` and where it saves its executable file. Remember to adjust the file ending according to what the `Makefile` outputs.

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "gcc - Build and debug active file",
            "type": "cppdbg",
            "request": "launch",
            "program": "${workspaceFolder}/bin/${fileBasenameNoExtension}.out",
            "args": [],
            "stopAtEntry": false,
            "cwd": "${workspaceFolder}",
            "environment": [],
            "externalConsole": false,
            "MIMode": "gdb",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                }
            ],
            "preLaunchTask": "build",
            "miDebuggerPath": "/usr/bin/gdb",
            "sourceFileMap": {
                "/build/glibc-2ORdQG": "/usr/src/glibc"
            }
        }
    ]
}
```

Now when debugging and the debugger quits the program, there will always be an error about now able to open a specific file such as `/build/glibc-2ORdQG` or some other letters and numbers after `glibc-...`. This is not a problem more than that it is annoying. This can be fixed by downloading the files which it wants to open.

15. Download `glibc` compressed file with `sudo apt install glibc-source`.
16. Go to the right directory `cd /usr/src/glibc`
17. Extract the content of the compressed file with `sudo tar xf glibc-2.27.tar.xz`
18. Now add the following, except the most outer curly brackets, to the `launch.json` file under `"configurations": [{...}]`
    - The letters and numbers `<LetterCombination>` after `glibc-...` must be adjusted to the error message that pops up when the debugger is quitting the program.

```
{
    "sourceFileMap": {
        "/build/glibc-<LetterCombination>": "/usr/src/glibc"
    }
}
```

2.1.3 Global multiple word search

Some times you might want to find a specific file or line of code with multiple words in it, without having to be in a direct sequence. Use this extension which automates the process of using `regex`.

Search by Alexander:

https://marketplace.visualstudio.com/items?itemName=usernamehw.search

2.2 LaTeX

Install the complete version TeX Live:

```
sudo apt install texlive-full
```

In Visual Studio Code, install the extension: `LaTeX Workshop`

2.2.1 Latex Workshop

Open up the settings `JSON` file:

`Ctrl + Shift + P` and enter

```
>Preferences: Open Settings (JSON)
```

Some times, the recipes does not appear in the `JSON` file. Append the code below if it doesn't exist.

```
    "latex-workshop.view.pdf.viewer": "browser",
    "latex-workshop.latex.autoBuild.run": "onSave",
    "latex-workshop.latex.autoClean.run": "onBuilt",
    "latex-workshop.latex.recipe.default": "first",
    "latex-workshop.chktex.run": "onType",
    "latex-workshop.chktex.delay": 2000,
    "latex-workshop.latex.recipes": [
        {
            "name": "latexmk",
            "tools": [
            "latexmk"
            ]
        },
        {
            "name": "pdflatex - bibtex - pdflatex x2",
            "tools": [
            "pdflatex",
            "bibtex",
            "pdflatex",
            "pdflatex"
            ]
        }
    ],
    "latex-workshop.latex.tools": [
        {
            "name": "latexmk",
            "command": "latexmk",
            "args": [
```

```
            "-synctex=1",
            "-interaction=nonstopmode",
            "-file-line-error",
            "-pdf",
            "-shell-escape",
            "-outdir=%OUTDIR%",
            "%DOC%"
            ],
            "env": {}
        },
        {
            "name": "pdflatex",
            "command": "pdflatex",
            "args": [
            "-synctex=1",
            "-interaction=nonstopmode",
            "-file-line-error",
            "%DOC%"
            ],
            "env": {}
        },
        {
            "name": "bibtex",
            "command": "bibtex",
            "args": [
            "%DOCFILE%"
            ],
            "env": {}
        }
    ]
```

Default PDF viewer

The setting

`"latex-workshop.view.pdf.viewer": "browser"`

will open up a browser tab when pressing the `View LaTeX PDF file` button. Using the browser works good with `Synctex` for jumping between PDF and code seamlessly.

Synctex - Jump between PDF and code

By having the flag `"-synctex=1"` in the recipe, one can enable jumping between the PDF and code locations seamlessly. Works good when using `"browser"` as default PDF viewer.

- `Ctrl` clicking in the PDF.
- Set text marker in code and press `Ctrl + Alt + J`

Default recipe

The setting

`"latex-workshop.latex.recipe.default": "first"`

will run the first/top recipe given in `"latex-workshop.latex.recipes":`. It can be changed to `lastUsed` but might confuse some times.

Make sure that `latexmk` is the first item.

Clean aux files

The setting

```
"latex-workshop.latex.autoClean.run": "onBuilt"
```

will remove the `aux` files generated from the compilation after that the compilation is done.

Chktex - Linting

The settings

```
"latex-workshop.chktex.run": "onType"
"latex-workshop.chktex.delay": 2000
```

will enable linting using `Chtex`, checking two seconds after stopped writing. Will show problems in the `Problems` tab.