Computer Setup Programming

Robin Hellmers

November 7, 2021

Contents

1 Windows Subsystem for Linux 2 (WSL2)

1.1 Installation

Installation method of WSL depends on which windows version you are running.

Check your Windows version and build number by pressing
`Win + R` → Entering `winver`

Depending on the version you have, see the below sections.

If you have:

- Windows version 2004 and higher, with build 1904 and higher

Then you can do the **Automatic installation** section.

If that is not the case, then you must at least have:

- Windows version 1903 or higher, with build 18362 or higher

If so, then if running the version

- 1903, the build number must be 18362.1049 or higher
- 1909, the build number must be 18363.1049 or higher

If any of those is true for your system, then you can do the **Manual installation** section.

1.1.1 Automatic installation

A simple command in PowerShell: `wsl --install`

1.1.2 Manual installation

Enable Windows Subsystem for Linux

Open PowerShell as Admin and run:
`dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart`

Enable Virtual Machine feature

Open PowerShell as Admin and run:
`dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart`

Restart your computer.

Download the Linux kernel update package

Download the latest package through **this link**.

Set WSL 2 as the default version

Every new distribution installed should run with the WSL version set as default.

Open PowerShell and run:
`wsl --set-default-version 2`

1.2 Distribution installation

1.2.1 Downloading and installing

First we are going to download the Ubuntu distribution with some PowerShell commands.

Open PowerShell and go to somewhere in your main drive where you want to install Ubuntu:
When unzipping later, a directory named Ubuntu will be created.
`cd C:\Users\<username>\`

The download is done with the command:
`Invoke-WebRequest -Uri <distro url> -OutFile ./Ubuntu.zip -UseBasicParsing`

The `<distro url>` must be chosen depending on which distribution you want *(Recommending 18.04)*:

- Ubuntu 20.04 **https://aka.ms/wslubuntu2004**
- Ubuntu 18.04 **https://aka.ms/wsl-ubuntu-1804**

Unzip:
`Expand-Archive ./Ubuntu.zip`

Close down PowerShell.

Open the File Explorer at the location which you have the unzipped files e.g. `C:\Users\<username>\Ubuntu\`.

Start the installation by running the `.exe` file e.g. `ubuntu2004.exe`.

1.2.2 Setup of the distribution

When installing, you will be prompted for a username.

If the username can't be chosen

If the username can't be chosen because of using e.g. a dot (.), press `Ctrl + C` and the window should close.

Run the `.exe` file again and you should now have skipped the first user initialization and be logged in as `root`.

```
adduser -aG --force-badname <username>
```

Set a password and you can skip entering all of your details by just pressing enter till it's done.

Now we are going to give the new user `sudo` privileges by adding it to the `sudo` group:

```
usermod -aG sudo <username>
```

Verify that the new user is in the `sudo` group with: `groups <username>`

Which should show: `<username> : <username> sudo`

Log into the new user:

```
su - <username>
```

Setting the default user

In order to set the default user to the newly created one, run the line below but replace `<username>`:

```
sudo bash -c 'printf "[user]\ndefault =<username>\n " >> /etc/wsl.conf'
```

Verify that the file has been created with the contents:

```
cat /etc/wsl.conf
```

Close the Ubuntu terminal.

Open up the Command Prompt `cmd` and see if the Ubuntu distribution is still running:

```
wsl -l -
```

If the `STATE` is `Running`, then terminate the session with:

```
wsl -t <distro name>
```

Verify that it has `Stopped`:

```
wsl -l -v
```

Close the `cmd`.

Then open up Ubuntu again with the `.exe`. Now it should be logged in with the new user, instead of `root`.

1.2.3 Update the system

Update the list of packages, but doesn't install:

```
sudo apt update
```

Install new version of packages and say yes to every question:

```
sudo apt upgrade -y
```

1.2.4 Set WSL to version 2

We are now going to check which WSL version which is used with the newly installed Ubuntu distribution.

Close any Ubuntu terminal open.

Open up the Command Prompt `cmd` and run:

```
wsl -l -v
```

Which should show all WSL distributions and which version they are running.

If your Ubuntu WSL version is 1, close any Ubuntu window and then run:

```
wsl --set-version <distro name> 2
```

After it is done, recheck that the version now is 2:

```
wsl -l -v
```

1.3 Relevant programs & setups

1.3.1 Windows terminal

Windows Terminal is a pretty new tool released from Microsoft in which you can do a lot of customizations such as:

- Use tabs for the same or different distributions
- Change colors
- Fast start SSH to server through Ubuntu
- And much more...

Installing

In order to continuously get updates, install winget by going to the **Microsoft Github page** and downloading a file called something like:

```
Microsoft.DesktopAppInstaller_8wekyb3d8bbwe.msixbundle
```

Execute the file and install/update.

Thereafter, open up `cmd` and enter `winget` to verify that it is accessible.

Open `cmd` and install Windows Terminal with:

```
winget install --id=Microsoft.WindowsTerminal -e
```

Custom colors

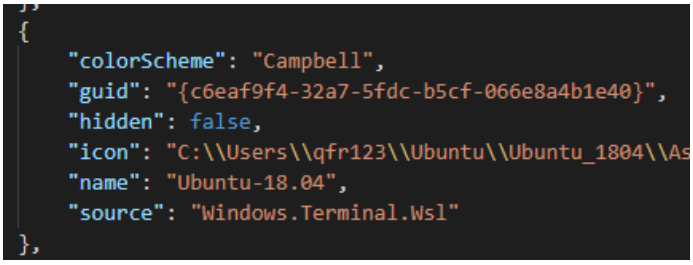
You are able to customize your colors in Windows terminal through a `.json` file. It is recommended setting Visual Studio Code as your standard `.json` file viewer as it shows a small color sample of every `##rrggbb` color.

Open Windows Terminal, and press `Ctrl + ,`. Then at the far bottom left corner, click **Open JSON file**.

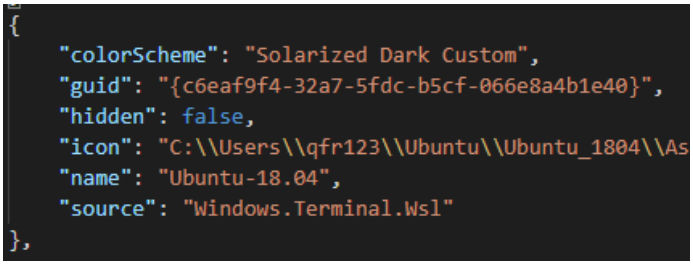
Here you can add custom schemes and do other settings. I recommend using my custom color scheme by copy and appending this to your schemes:

```
{
    "name": "Solarized Dark Custom",
    "background": "#002B36",
    "black": "#002B36",
    "blue": "#FFFFFF",
    "brightBlack": "#073642",
    "brightBlue": "#747D7D",
    "brightCyan": "#93A1A1",
    "brightGreen": "#00A0FF",
    "brightPurple": "#6C71C4",
    "brightRed": "#CB4B16",
    "brightWhite": "#FDF6E3",
    "brightYellow": "#657B83",
    "cursorColor": "#FFFFFF",
    "cyan": "#2AA198",
    "foreground": "#A8A8A8",
    "green": "#056E32",
    "purple": "#D33682",
    "red": "#DC322F",
    "selectionBackground": "#FFFFFF",
    "white": "#EEE8D5",
    "yellow": "#B58900"
},
```

Then, within the `json` file, go to profiles and look for your Ubuntu distribution.



Change `"colorScheme"` to `"Solarized Dark Custom"`

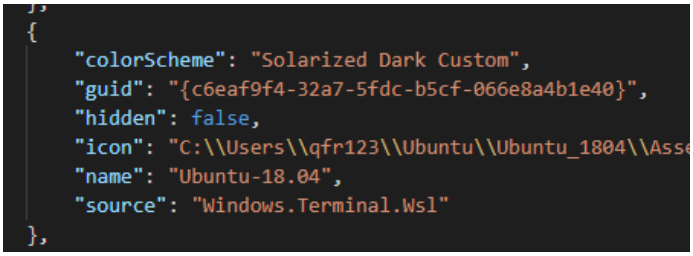


Set your distro as default

Open the settings `.json` file through `Ctrl + ,` then pressing **Open JSON file** far bottom left.

You can choose the default distribution to be opened up when opening Windows Terminal.

Change `<guid>` in `"defaultProfile": <guid>` to the one corresponding to the profile with your distribution.



Default location

Also, one might want Ubuntu to open up at a specific location such as the Ubuntu home folder.

In Ubuntu:

```
cd ~ && explorer.exe .
```

Copy the path and add the key

```
"startingDirectory": <path>
```

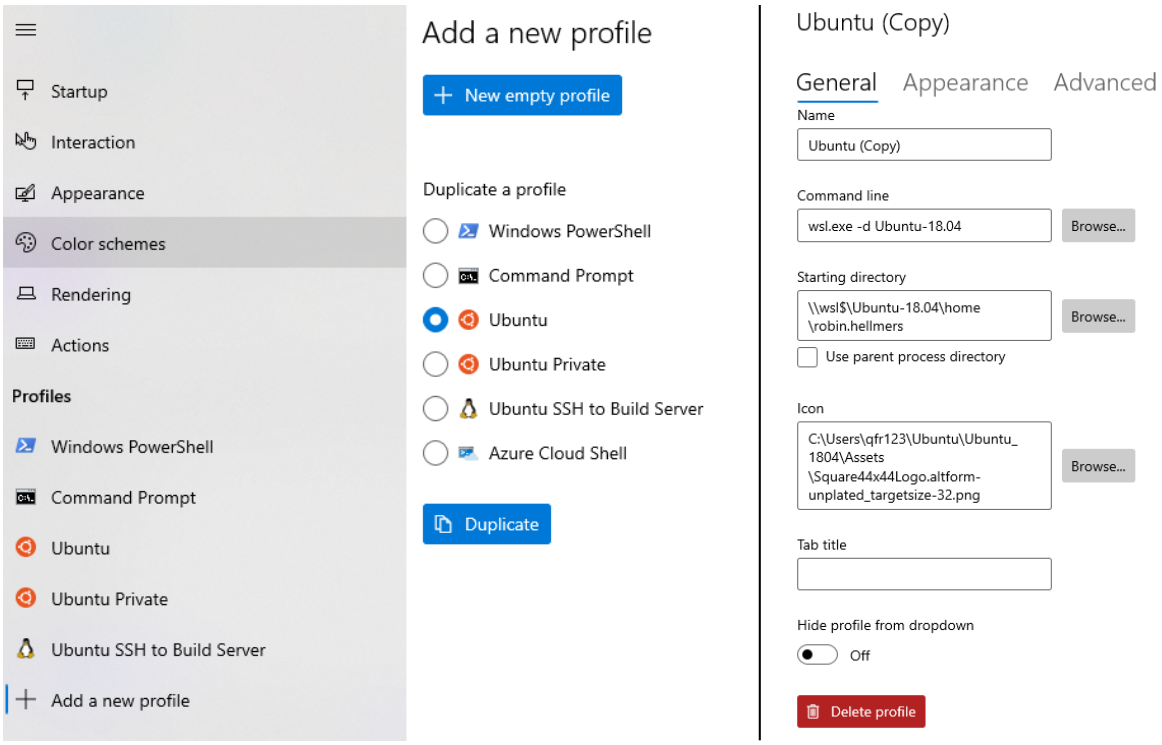
and replace the path. For every `\` you will need to add another `\` in order for windows to understand.

```
\\\\wsl$\\Ubuntu-18.04\\home\\<username>
```

Profile for SSH through distribution

In Windows Terminal, open up the settings with `Ctrl + ,`.

Press + **Add a new profile** and chose your distribution to duplicate. Press **Duplicate**:



Give the new name e.g. *Ubuntu SSH to Server*.

At the field **Command line**, append

```
ssh <username>@<IP address server>
```

or if your have a password for the server username

```
ssh-pass -p <password> ssh <username>@<IP address server>
```

so it looks something like

```
wsl.exe -d Ubuntu-18.04 ssh <username>@<IP address server>
```

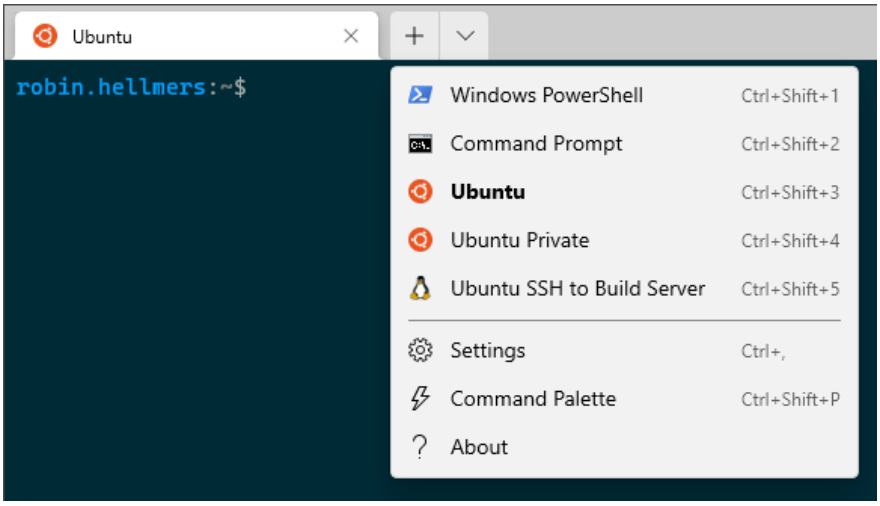
or

```
wsl.exe -d Ubuntu-18.04 ssh-pass -p <password> ssh <username>@<IP address server>
```

.

Then **Save** it.

Now press the newly added profile and see if you can SSH into the server.



If it prompts you for your password. Open up at normal Ubuntu window and enter

```
ssh-copy-id <username>@<IP address build server>
```

and enter your password. Now try to SSH again.

1.3.2 Docker for WSL2

In Windows:

Download Docker Desktop

Install and make sure that the following checkboxes are ticked:

- Enable Hyper-V Windows Features (Probably not necessary because of using WSL2 and not WSL1, but won't hurt)
- Install required Windows components for WSL 2

Run the Docker Desktop program in Windows. If the engine never starts, try to reboot your computer.

In Docker Desktop go into settings through the gear icon in the top right and then:

- **General** make sure that
  - The **Use the WSL 2 based engine** checkbox is checked.
- **Resources** → **WSL INTEGRATION** and make sure that
  - The **Enable integration with my default WSL distro** checkbox is checked.
  - Your installed distribution is enabled.

Open up Ubuntu and check if docker is available:

```
docker --version
```

Test docker connection between distribution and Windows

Time to test the connection between Windows docker and Ubuntu docker which both runs through WSL2.

In **Ubuntu terminal**, see which containers that exists:

```
docker ps -a
```

Which should be empty right now.

Open up `cmd` in **Windows** and run `hello-world`:

```
docker run hello-world
```

When it is done, see which containers that exists

```
docker ps -a
```

Which should show `hello-world` as `IMAGE` with a `CONTAINER ID`.

Go back to the **Ubuntu terminal** and see which containers that exists:

```
docker ps -a
```

Which should show `hello-world` as `IMAGE` with the same `CONTAINER ID` as in Windows.

1.3.3 Packages

trash-cli - Remove files with possibility to restore

Usually when using the `rm` command, it is very hard to restore what has been removed. By using this package, a trashbin is being used before removing permanently.

1. Install the `trash-cli` package with `sudo apt install trash-cli`
2. Then set the alias `alias rm=trash` in your `~/.bashrc` file.

The `trash-cli` package comes with some commands:

- `trash` - Which is like `rm` but it will be put in a trashbin instead
- `trash-list` - Lists everything in the trashbin
- `restore-trash` - Lists and numbers everything in the trashbin. Asks for an index from the list to restore.
- `trash-empty` - Permanently remove everything in the trashbin.

apparix - Terminal bookmark directories

Install the `apparix` package with `sudo apt install apparix`

Run `apparix` in order for it to set up its folders.

Then write `apparix --shell-examples` and copy everything except the aliases at the bottom. Paste this in `~/.bashrc`

If just copying from the terminal and pasting into `~/.bashrc` doesn't work, create a file and let the output be written into that instead in order to copy from it.

```
touch text.txt
```

```
apparix --shell-examples > text.txt
```

Then paste it into `/.bashrc`.

Restart console.

Bookmark current directory with `bm bookmarkname` and go to the same location with `to bookmarkname`

1.3.4 Extras

Git indications Shorten terminal path shown

If you navigate through many directories inside of each other, the terminal gets input line gets very long. This will shorten that to show only the latest 3 directories.

This will also show Git branch and indication about commits to be pushed etc.

1. Download the **.git-completion.bash** and save it as the very same name at `~/.git-completion.bash`. If you want to save it elsewhere, you have to change the first row of the code you are going to copy very soon.
2. Use `sudo chmod +x ~/.git-completion.bash` in order to give permission to the user to run it through `.bashrc`.
3. Copy the code:

```
~/.git-completion.bash

export PROMPT_DIRTRIM=3
PS1_custom='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u\[\033[00m\]:\['\
'\033[01;34m\]\w\[\033[00m\]\$ '
PS1_original='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m'\
'\]:\[\033[01;34m\]\w\[\033[00m\]\$ '

if [ "$color_prompt" = yes ]; then
    PS1=$PS1_custom
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
unset color_prompt force_color_prompt

export GIT_PS1_SHOWCOLORHINTS=true
export GIT_PS1_SHOWDIRTYSTATE=true
export GIT_PS1_SHOWUNTRACKEDFILES=true
export GIT_PS1_SHOWUPSTREAM="auto"
# PROMPT_COMMAND='__git_ps1 "\u@\h:\w" "\\\$ "'
# use existing PS1 settings
PROMPT_COMMAND=$(sed -r 's|^(.+)(\\\$\s*)$|__git_ps1 "\1" "\2"|' <<< $PS1)
```

4. Open up `~/.bashrc` and replace/paste the corresponding code similar to:

```
if [ "$color_prompt" = yes ]; then
    PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\\
033[01;34m\]\w\[\033[00m\]\$ '
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
unset color_prompt force_color_prompt
```

5. In Ubuntu, enter `source ~/.bashrc` or restart the terminal.

Fix directory highlight inconsistency

Depending if you have created the directories within Ubuntu WSL directly with `mkdir` or pulled a directory, which were made within Windows, through some version control system; the background or highlight coloring of these directory may vary.

1. Run `dircolors -p > ~/.dircolors` and open `~/.dircolors` in an editor.
2. Find `DIR`, `STICKY_OTHER_WRITABLE`, `STICKY` and write their corresponding values in the comments in case one want to revert the upcoming changes.
3. Find `OTHER_WRITABLE` and copy the variable value, which probably is `34;42`.
4. Replace the values of `DIR`, `STICKY_OTHER_WRITABLE`, `STICKY` with the copied value of `OTHER_WRITABLE`.
5. Restart the Ubuntu terminal.

Disable bell sound when using backspace

If you don't have anything to remove in the terminal and press backspace, you will hear a bell sound. Disable this by entering

```
echo 'set bell-style none' > ~/.inputrc
```

Restart the terminal.

2 Git

2.1 Multiple repositories

2.1.1 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>/.
```

2.2 Adding a remote

2.2.1 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`

Remember passphrase

By using `ssh-agent`, you do not have to enter the SSH-key passphrase everytime.

Append the following to the `~/.profile`

```
SSH_ENV="$HOME/.ssh/agent-environment"

function start_agent {
    echo "Initialising new SSH agent..."
    /usr/bin/ssh-agent | sed 's/^echo/#echo/' > "${SSH_ENV}"
    echo succeeded
    chmod 600 "${SSH_ENV}"
    . "${SSH_ENV}" > /dev/null
    /usr/bin/ssh-add;
}

# Source SSH settings, if applicable

if [ -f "${SSH_ENV}" ]; then
    . "${SSH_ENV}" > /dev/null
    #ps ${SSH_AGENT_PID} doesn't work under cywgin
    ps -ef | grep ${SSH_AGENT_PID} | grep ssh-agent$ > /dev/null || {
        start_agent;
    }
else
    start_agent;
fi
```

then `source ~/.profile` and the `ssh-agent` will ask for the passphrase and then you do not have to enter it again. The script will look for the agent automatically.

You can manually add a SSH-key with:

`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.

2.3 Configurations

2.3.1 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)'
```

2.3.2 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.

2.3.3 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
```

3 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>`.

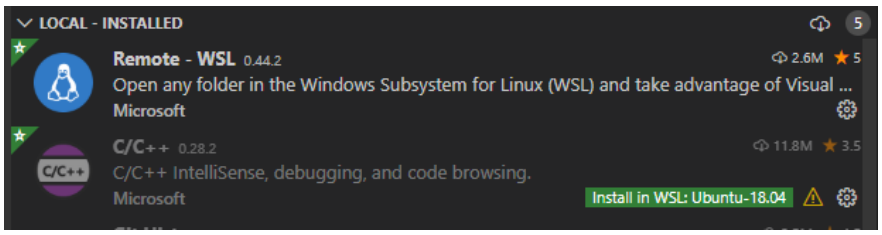
See Visual Studio Code's connection to the Ubuntu WSL by clicking on the **Remote Explorer Icon** to the left. Make sure that the dropdown menu at the top shows **WSL Targets**.

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**.

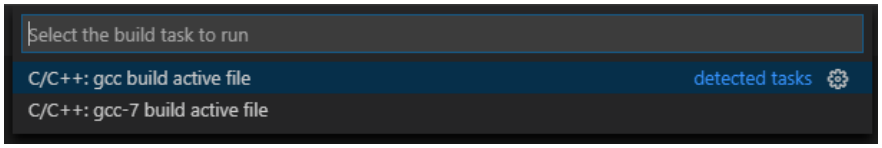
3.1 C programming

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.



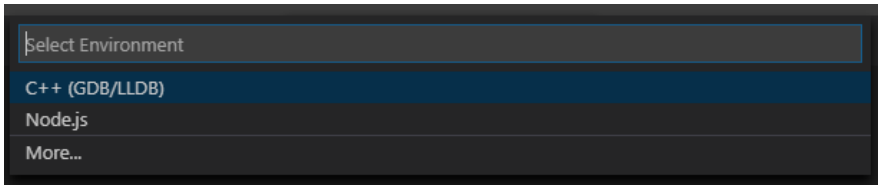
7. By now it might prompt that you have to reload the window. Press that button.
8. Open up the folder you created the main C file in. `File` → `Open Folder...`
9. Open up the `helloworld.c` file in the file explorer.
10. Press `Terminal` → `Configure Default Build Task...`. In the dropdown list that 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`.



3.1.1 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.

3.1.2 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`.

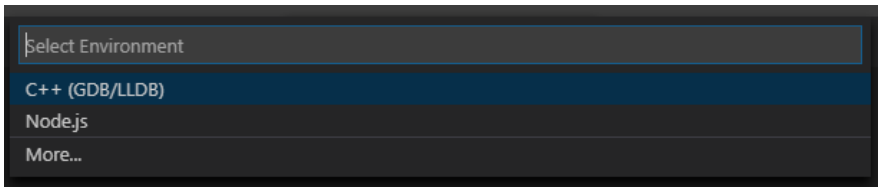
```
CC := gcc
CFLAGS := -pthread -g

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"
    }
}
```

3.1.3 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

3.2 LaTeX

Install the complete version TeX Live:

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

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

3.2.1 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.

3.2.2 Packages

Minted - Code viewing

In order to use the `minted` package for displaying code one must install `Pygments` for `Python`.

If using `Ubuntu 18.04`:

```
sudo apt install python-pygments
```

If using `Ubuntu 20.04`:

```
sudo apt install python3-pygments
```

Also, one must use the `-shell-escape` flag with the `latexmk` (standard compilation) command in order to compile with `minted`.

In Visual Studio Code, press `Ctrl + Shift + P` in order to search for commands. Search and execute `Preferences: Open settings (JSON)` in order to open up the `settings.json` file with the compilation recipe. Add `"-shell-escape"` to the arguments.

If the file is empty, use the `settings.json` from Github.

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

If there there is an error message similar to

`Undefined control sequence. \PYG #1#2->\FV@PYG`

WARNING: Slows down compilation significantly!

Add the argument `cache=false` when loading the `minted` package.

`\usepackage [cache=false]{minted}`

3.2.3 Extras

Glossaries

In order for glossaries to work, one must call `makeglossaries`.

E.g. `pdflatex <file>.tex` → `makeglossaries <file>` → `pdflatex <file>.text`

But with **`latex-workshop`** in `VSCode`, the standard compilation tool is `latexmk` which does a series of calls with e.g. `pdflatex`, `bibtex`, etc.

In order to make `latexmk` include a call of `makeglossaries`,

create the file `~/.latexmkrc` from Github.

```
add_cus_dep('glo', 'gls', 0, 'makeglo2gls');
add_cus_dep('acn', 'acr', 0, 'makeglo2gls');
sub makeglo2gls {
        system("makeglossaries $_[0]");
}
```

3.3 Extras

3.3.1 Word wrap - Continue code on next line

You can use word wrap as default in Visual Studio Code settings by pressing `Ctrl + ,` and searching for

`editor.wordWrap`

Or you can change it for the current file by pressing `Ctrl + Shift + P` and searching for

`View: Toggle Word Wrap`

Or edit the shortcut key at

`File` → `Preferences` → `Keyboard Shortcuts`