Computer Setup Programming

Robin Hellmers

June 19, 2021

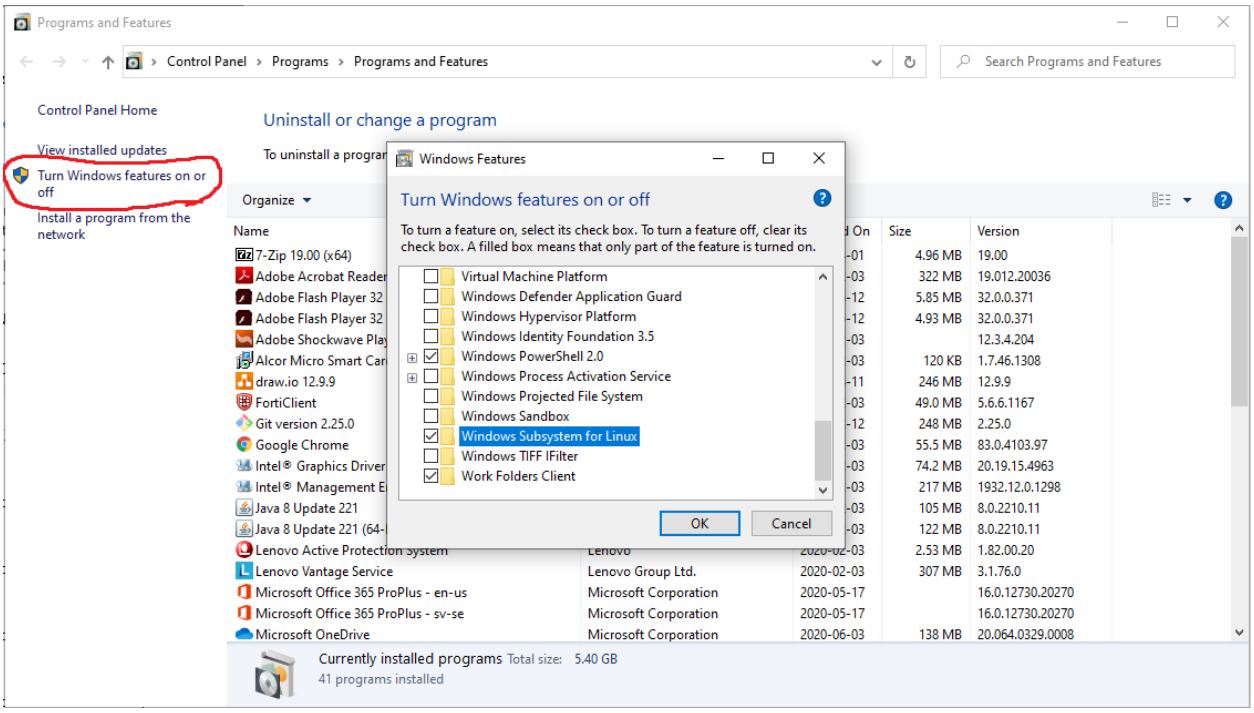
Contents

1 Ubuntu App - Windows Subsystem for Linux

1.1 Installation Ubuntu WSL with Visual Studio Code

`https://code.visualstudio.com/docs/cpp/config-wsl`

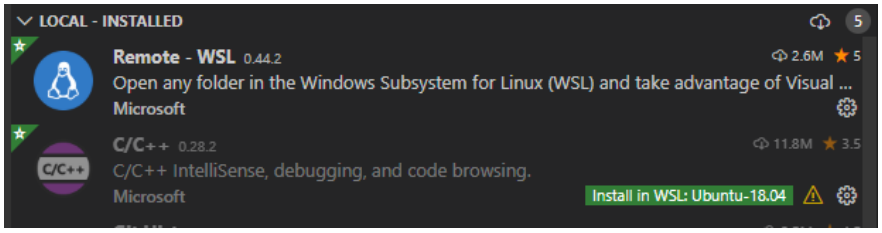
1. Download Ubuntu 18.04 LTS from Windows Store.
2. Activate `Windows Subsystem for Linux` through `Programs and Features`.



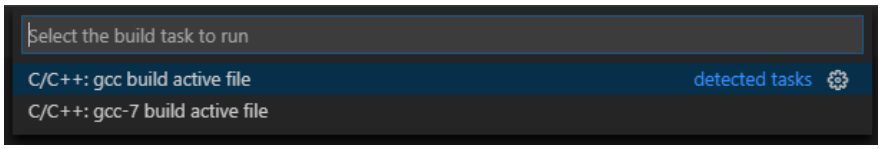
3. Restart computer.
4. Run `Ubuntu 18.04 LTS` and let it install. Might have to press `enter` after a while.
5. Create user with password.
6. Install `gcc` and `gdb` on the Windows Subsystem for Linux (WSL).
   (a) Run `sudo apt update`
   (b) Then `sudo apt install build-essential` for `gcc`
   (c) Then `sudo apt install gdb` for `gdb`
7. 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.
8. Open up Visual Studio Code.
9. Install two extensions in VS code:

- `C/C++` from Microsoft
- `Remote - WSL` from Microsoft

10. Press on the new icon on the left, `Remote explorer`. Right-click the `Ubuntu 18.04` and press `Connect to WSL`. A new window will appear with some connection to the WSL.
11. Press on the extension icon the left in the new window. Press `Install in WSL: Ubuntu-18.04` button on the `C/C++` extension.

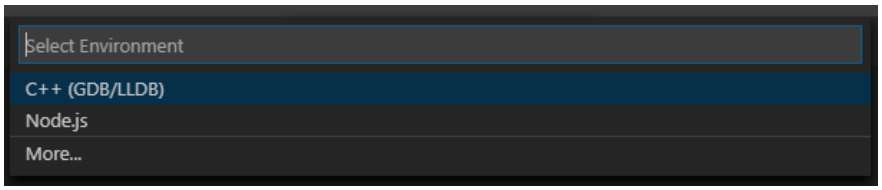


12. By now it might prompt that you have to reload the window. Press that button.
13. Open up the folder you created the main C file in. `File->Open Folder...`
14. Open up the `helloworld.c` file in the file explorer.
15. Press `Terminal->Configure Default Build Task...`. In the dropdown list that should 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`.



1.1.1 Simple single file compilation

16. Do not edit the `tasks.json`
17. 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.
18. 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.



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

1.1.2 Multi-file compilation with Makefile

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

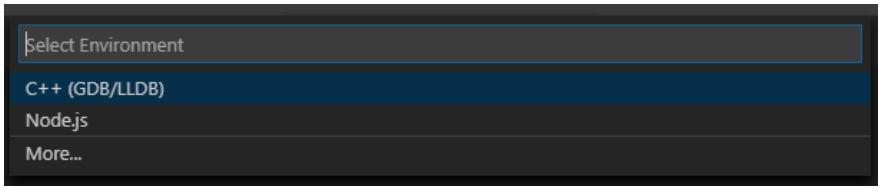
After creating one for the specific project, continue with the **Visual Studio Code** configuration:

16. 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/computer_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"
        }
    ]
}
```

17. 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.
18. 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.



19. 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/computer_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.

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

```
{
    "sourceFileMap": {
        "/build/glibc-2ORdQG": "/usr/src/glibc"
    }
}
```

1.1.3 Extra

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

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

1.2 WSL Ubuntu installations

Start with:

`sudo apt update`

1.2.1 Terminal bookmark directories

Install `Apparix` (Doc. `https://www.micans.org/apparix/`) with
`sudo apt-get 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`

1.2.2 Compiler `gcc` & `g++`

Compiler gcc and g++ installation:

```
sudo apt install build-essential
```

1.2.3 Debugger `gdb`

```
sudo apt install gdb
```

1.2.4 Git

Git installation:

```
sudo apt install git
```

Add credentials that will be asked for later on if not done:

```
git config -global user.email "your@email.com"
```

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

Initialize local repository:

```
git init
```

```
git remote add origin <remote-address>
```

Save credentials:

```
git config credential.helper store
```

Get master from remote origin:

```
git pull origin master
```

In order to not have to specify `<remote>` and `<branch>` in `git pull <remote> <branch>`, but still have to do previous pull first:

```
git branch -set-upstream-to=origin/master master
```

```
git pull
```

In order to pull another branch on the remote, firstly fetch the branches:

```
git fetch
```

Then see which branches that are fetched e.g.

```
* [new branch]        testing   -> origin/testing
```

Checkout that branch. OBS that you don't have a branch created locally, you will only be checking out the remote branch which may change as soon as you `git fetch` again. This is only in order to copy the content of the remote branch into the local one.

```
git checkout origin/testing
```

Create the local copy of the remote branch and go to it:

```
git checkout -b testing
```

Make sure that the local branch have its upstream to the remote one:

```
git branch -set-upstream-to=origin/testing
```

1.2.5 LaTeX

Install the complete version TeX Live:

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

1.2.5.1 Minted - Code

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`, add the argument `cache=false` when loading the `minted` package.

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

1.3 Ubuntu terminal colors

1.3.1 Everything step-by-step

A top-down list of everything in section 1.3.2, 1.3.3 and 1.3.4. Concentrating on what to actually do, without much explanation.

a) Get ColorTool from Microsoft in order to set a theme.

1. Go to `https://github.com/microsoft/terminal/tree/master/src/tools/ColorTool`
2. Click the link under the title **Installing** in `README.md` in order to see the latest release.
   E.g. `https://github.com/microsoft/terminal/releases/tag/1904.29002`
3. Download the ColorTool zip file.

b) Activate a theme with ColorTool.

1. Open `Command Prompt` in Windows.
2. In `CMD` look at the PATH variable with `PATH` or `echo %PATH%`
3. Copy one of the locations shown to store the ColorTool and color scheme.
   E.g. `C:\Users\Robin.Hellmers\AppData\Local\Microsoft\WindowsApps`
4. Unzip the files in that location.
5. Go to the very same directory in `CMD` with
   E.g. `cd C:\Users\Robin.Hellmers\AppData\Local\Microsoft\WindowsApps`
6. In the `CMD`, run `ColorTool -b solarized_dark.itermcolors`.
7. Restart the `Ubuntu` terminal.

c) Fix directory highlight inconsistency (Different permissions for `Windows` and `Ubuntu`).

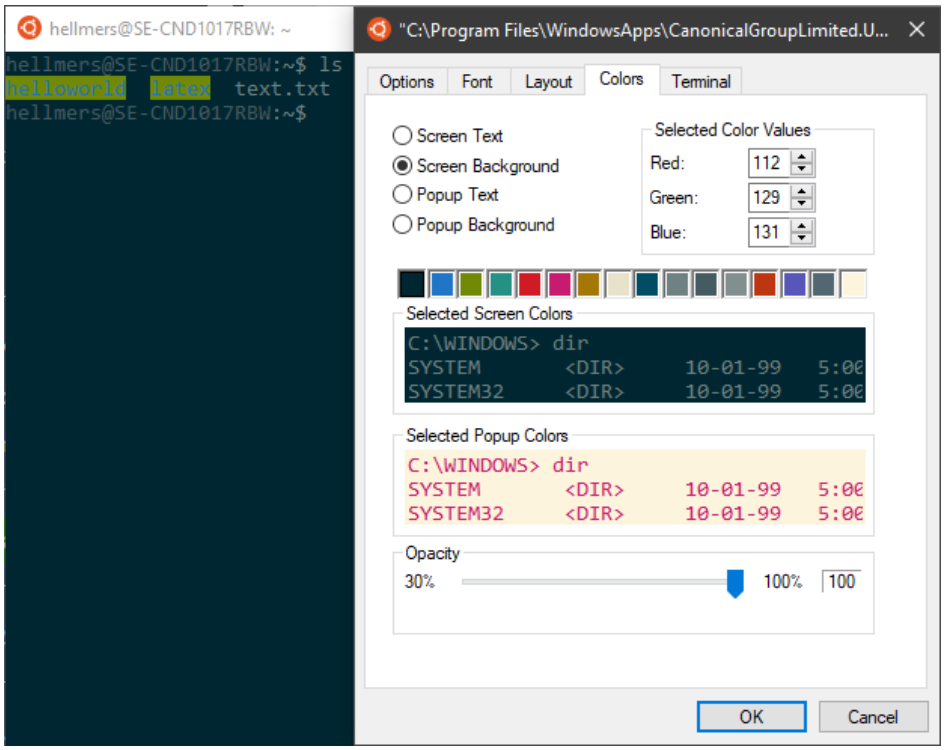
1. In the `Ubuntu` terminal, run `dircolors -p >  /.dircolors`
2. Find `DIR`, `STICKY_OTHER_VARIABLE`, `STICKY` and write their corresponding values in the comments in case one want to revert the upcoming changes.
3. Find `OTHER_WRITABLE` and copy the variable value e.g. `34;42`.
4. Replace the values of `DIR`, `STICKY_OTHER_VARIABLE`, `STICKY` with the copied value.
5. Restart the `Ubuntu` terminal.

d) Set specific colors with the properties tool

- BE CAREFUL, weird tool
- In order to revert back
  a) Open the `Registry editor (regedit)` and
  remove `HKEY_CURRENT_USER\Console\something_with_ubuntu_in_the_name`.
  b) Open up **Properties** and the **Colors** tab. Click `OK` to recreate the registry which you removed.

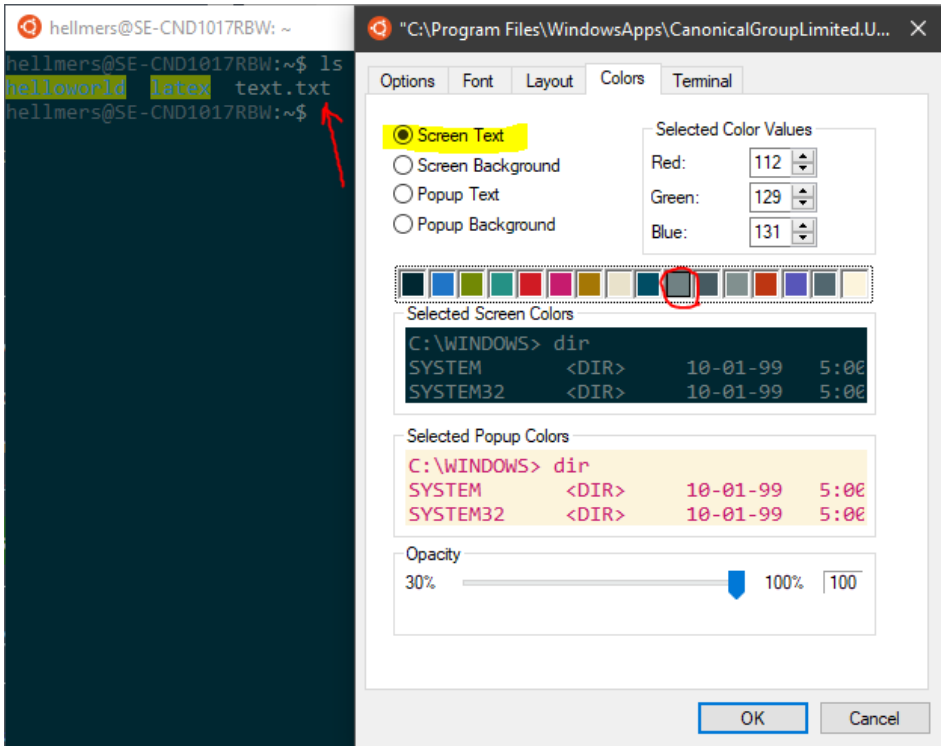
1. Open the `Ubuntu` terminal.

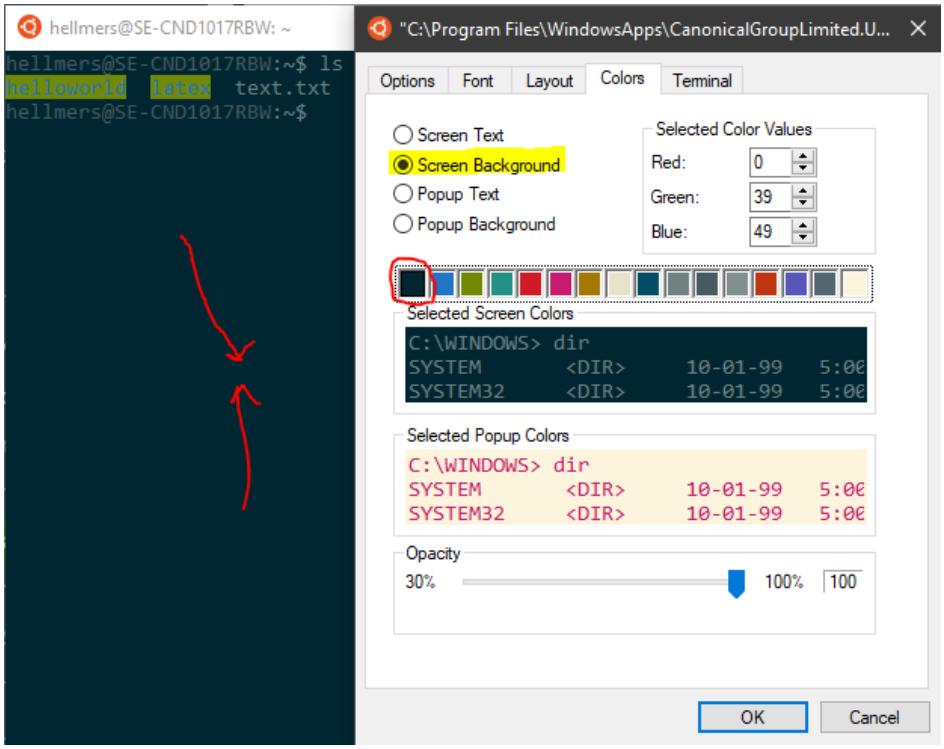
2. Right-click on the top bar which says e.g. **Ubuntu 18.04 LTS** to the right of the icon logo. Press **Properties** and go to the **Color** tab.



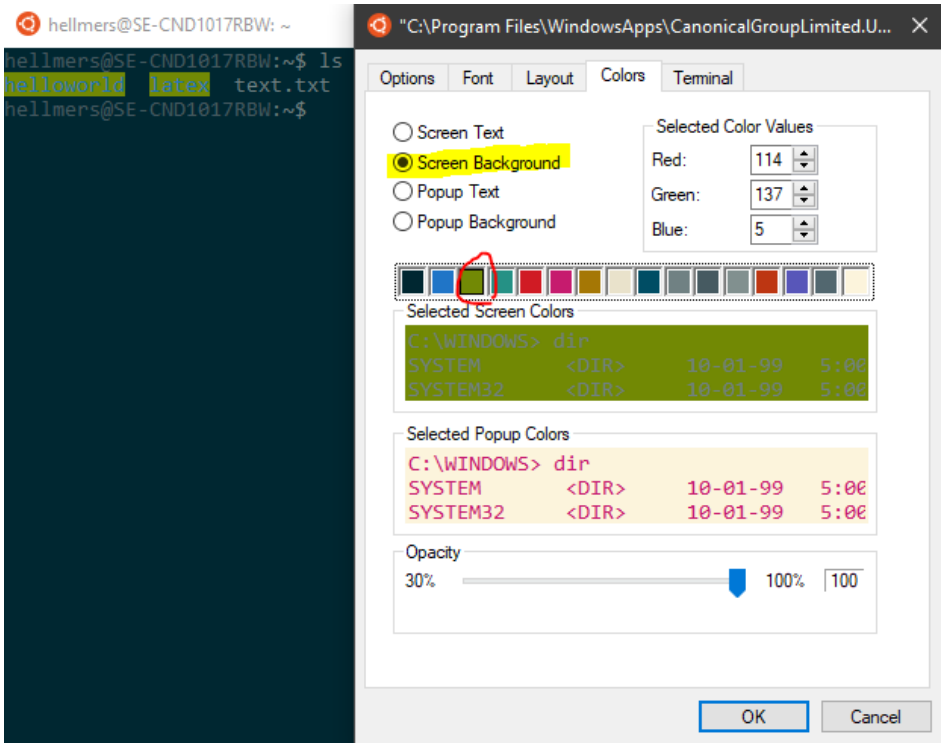
3. Click between **Screen Background** and **Screen Text** in order to see which colors are highlighted and chosen as standard for each. Remember which is highlighted for which (main colors).

   - In order to see the RGB values, you must click on the highlighted color. But beware that the color which you select must be the highlighted one for that specific category e.g. **Screen Background** or else you will change it.

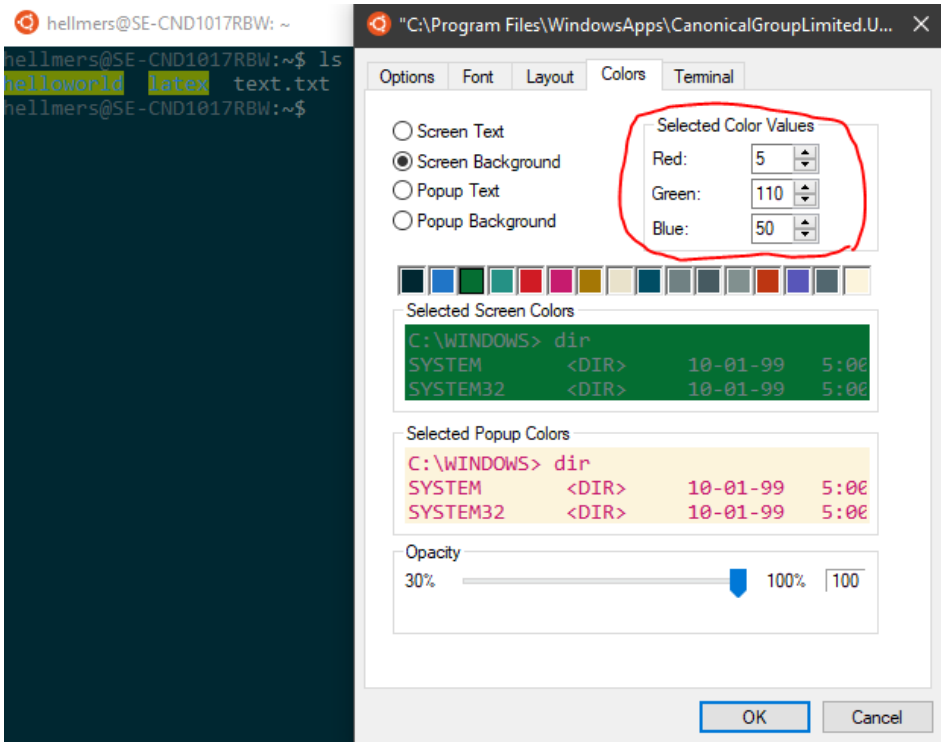




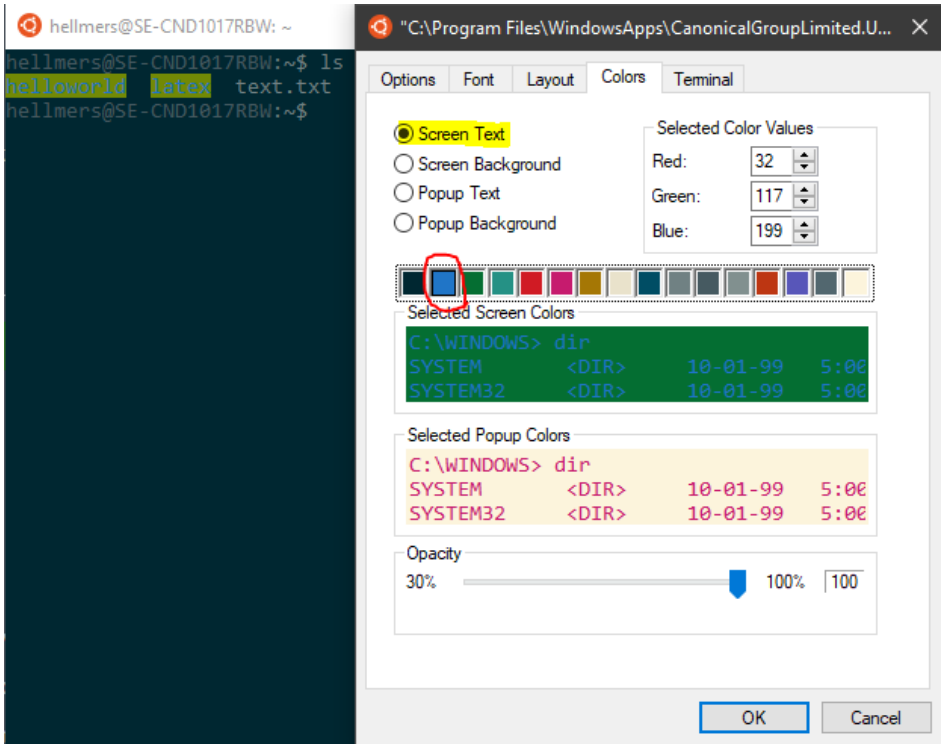
4. Choose **Screen Background** and select the 3rd color (green).



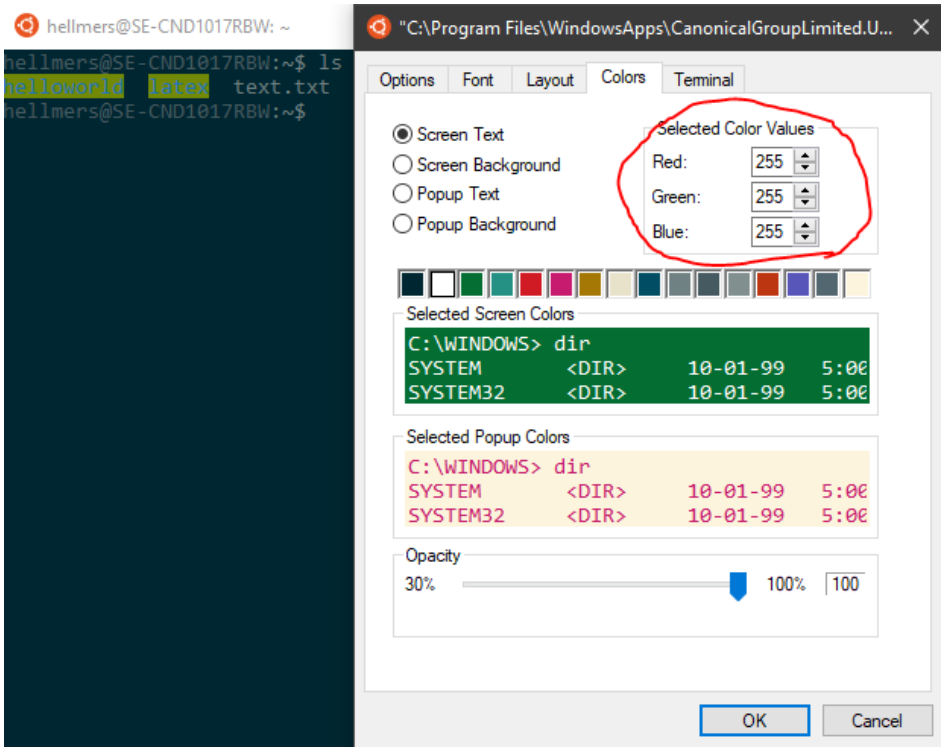
5. Replace the RGB colors to the same as the figure below.



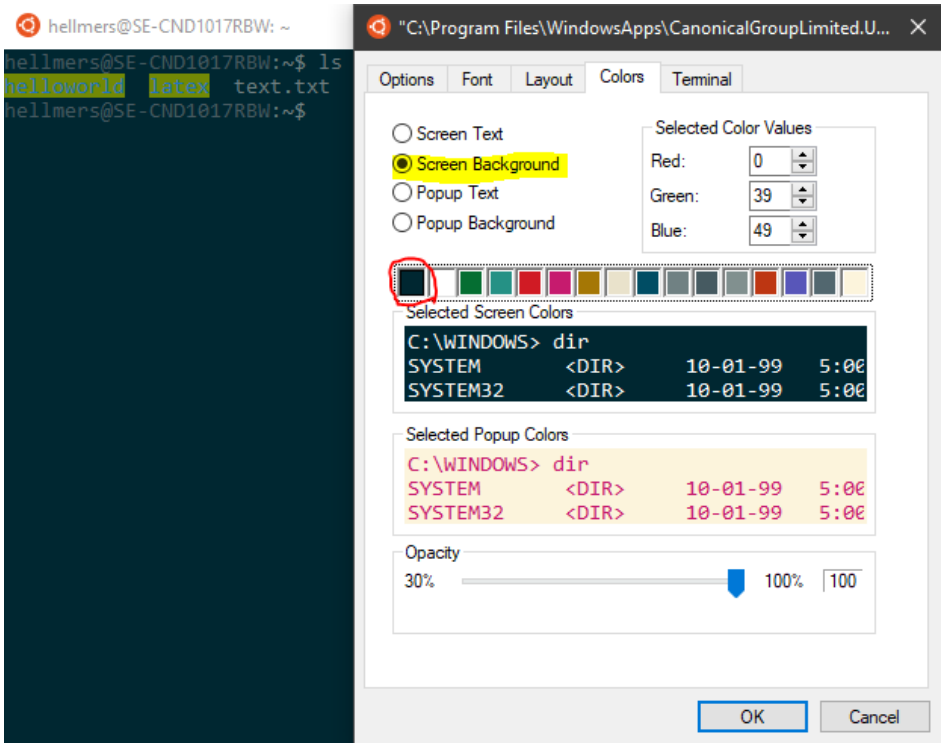
6. Choose **Screen Text** and select the 2nd color (blue).



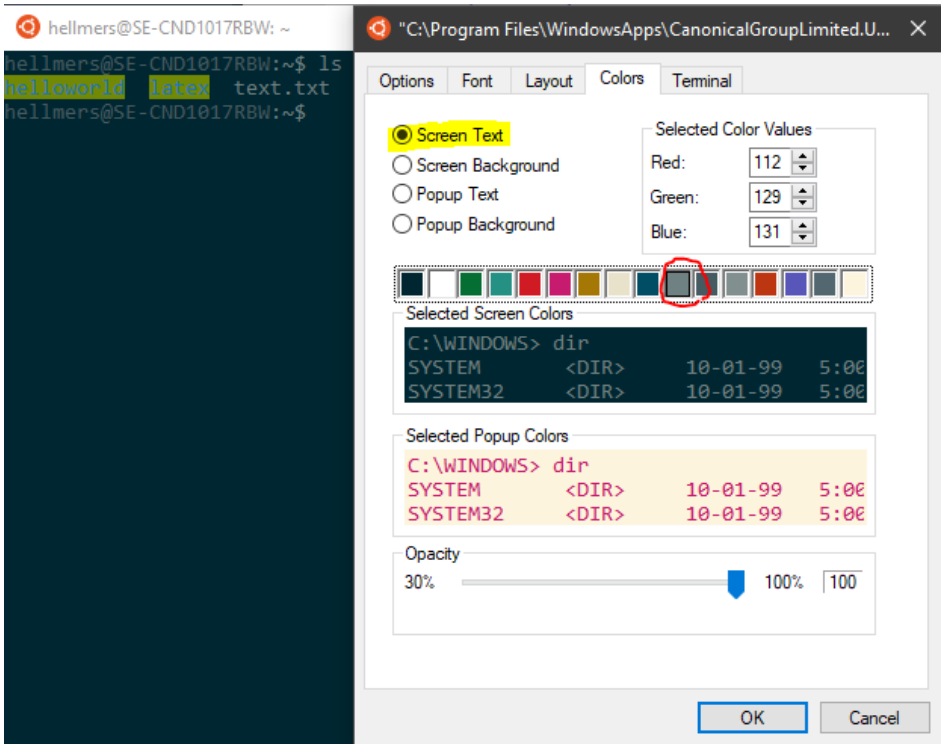
7. Replace the RGB colors to the same as the figure below.



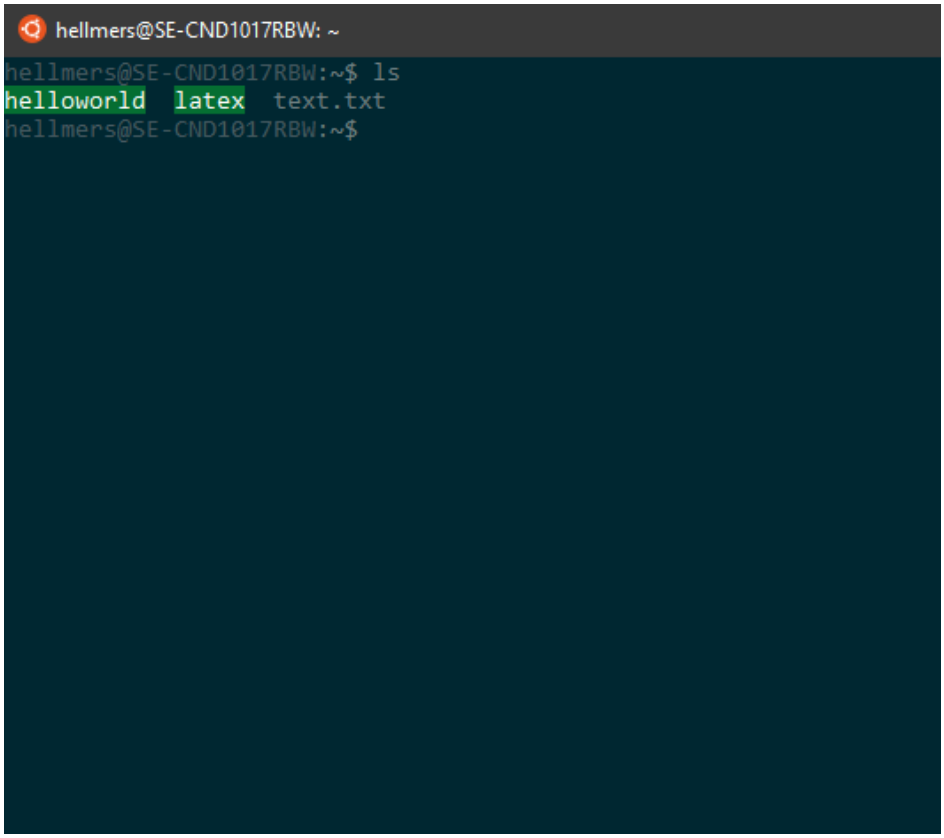
8. Choose **Screen Background** and select the main color for **Screen Background**.



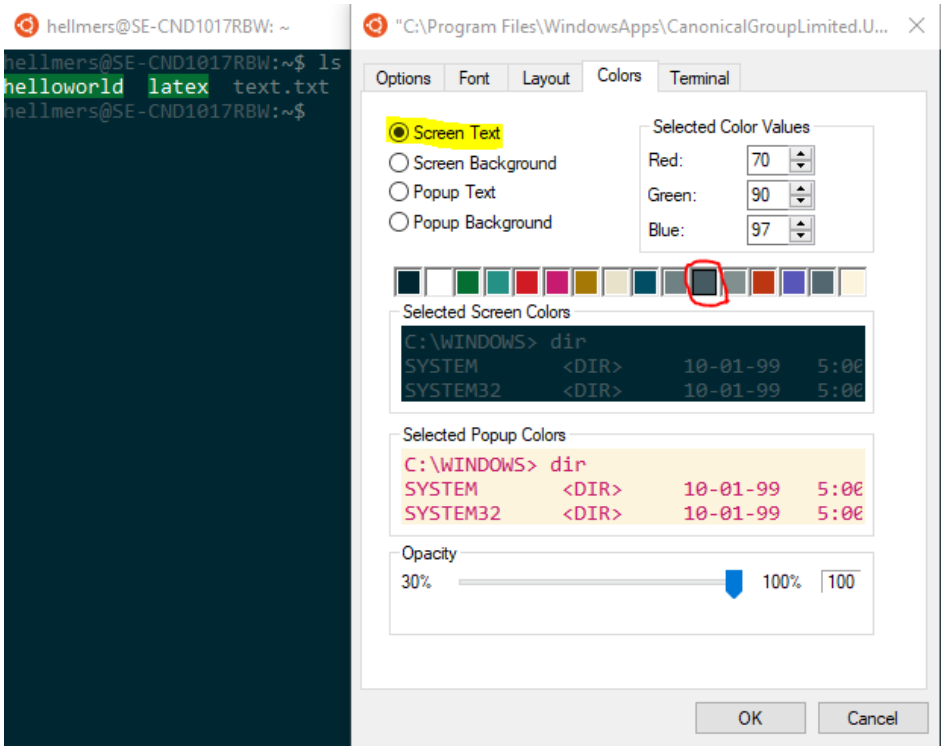
9. Choose **Screen Text** and select the main color for **Screen Text**.



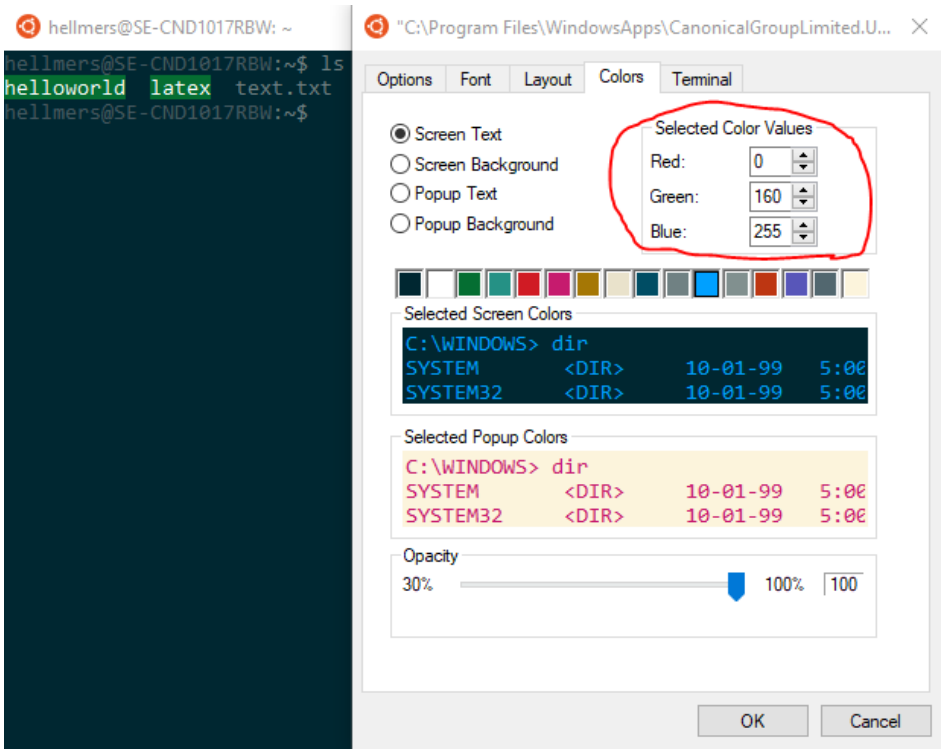
10. Press `OK` to save the changes.



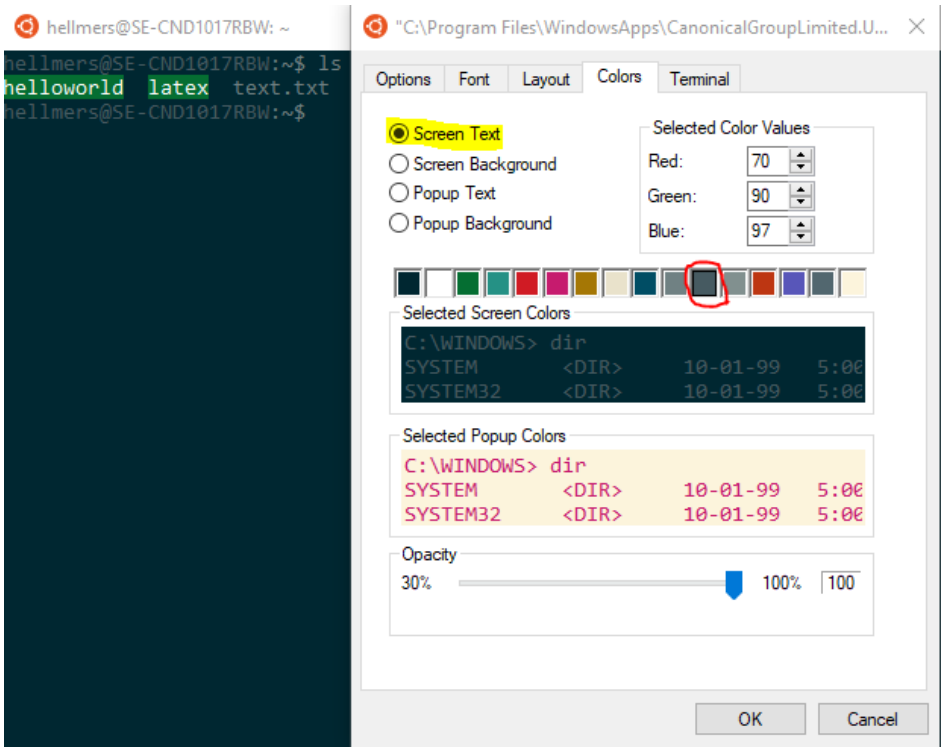
11. Open up **Properties** and the **Colors** tab again.
12. Choose **Screen Text** and select the 11th color (grey).



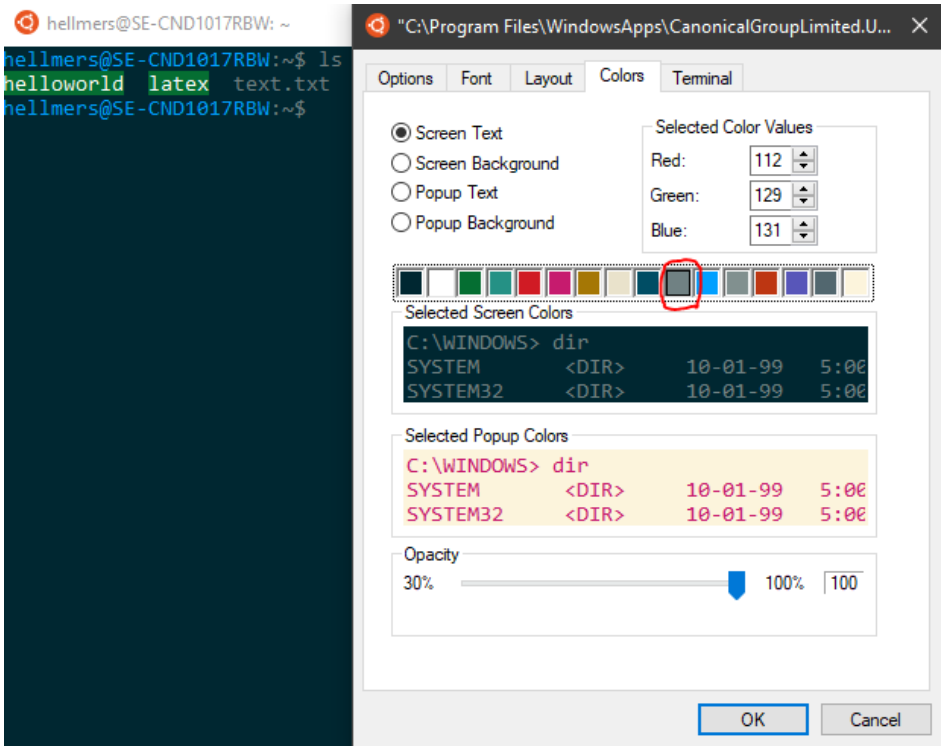
13. Replace the RGB colors to the same as the figure below.



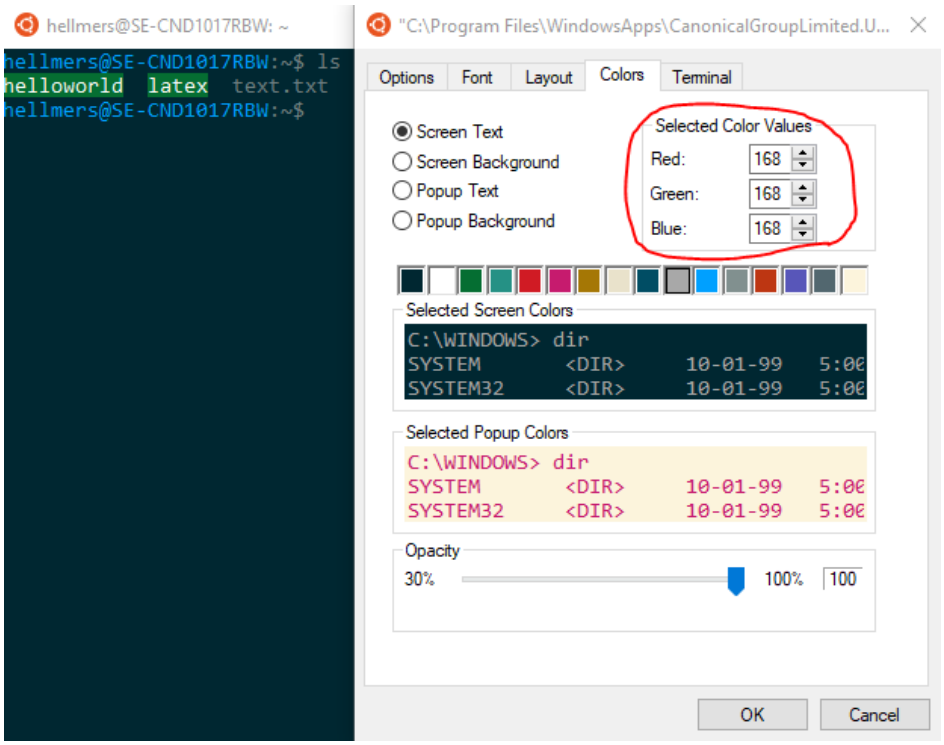
14. Choose **Screen Text** and select the 11th color (grey).



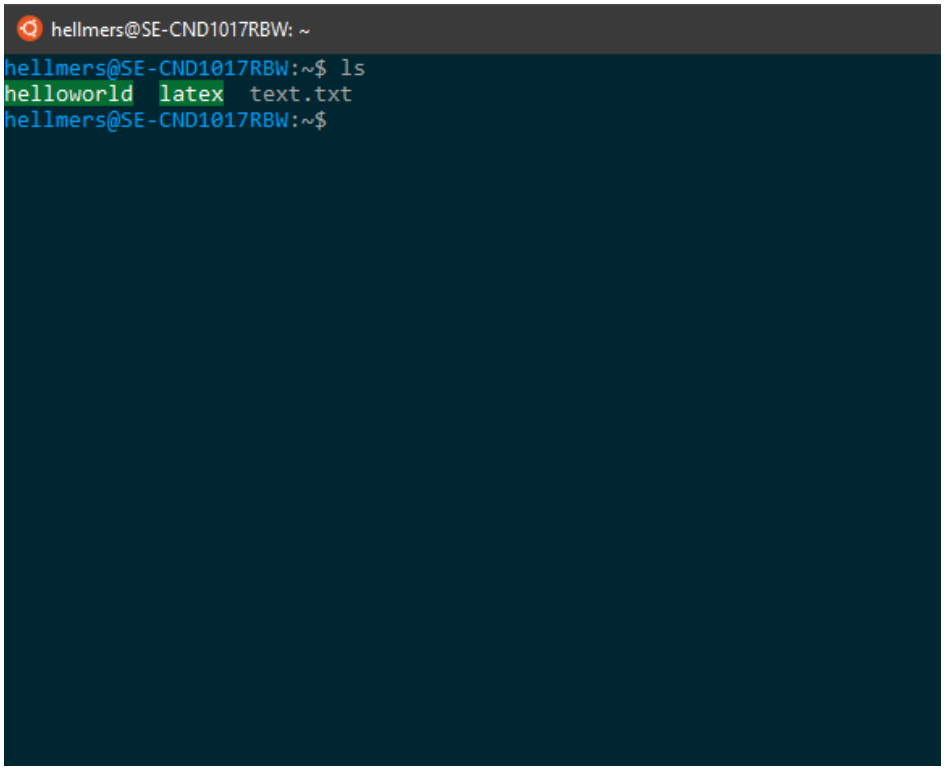
15. Select the main color for **Screen Text**.



16. Replace the RGB colors to the same as the figure below.



17. Click `OK` to save the changes.



e) Change `vimdiff` colors

1. Create the directory `~/.vim/colors/`.
2. Create the file `~/.vim/color/mycolorscheme.vim`
3. Paste this into the file (see Github):

```
highlight DiffAdd    cterm=bold ctermfg=15 ctermbg=22 gui=none guifg=bg guibg=Red
highlight DiffDelete cterm=bold ctermfg=15 ctermbg=88 gui=none guifg=bg guibg=Red
highlight DiffChange cterm=bold ctermfg=15 ctermbg=17 gui=none guifg=bg guibg=Red
highlight DiffText   cterm=bold ctermfg=15 ctermbg=130 gui=none guifg=bg guibg=Red
```

4. Create the file `~/.vimrc`
5. Paste this into the file (see Github):

```
if &diff
    colorscheme mycolorscheme
endif
```

6. Now the custom color scheme should be applied every time you open `vimdiff`.

1.3.2 Basic colors with Colortool

Colors in the Ubuntu App can be bad. Microsoft have released `ColorTool` to fix this. See the link below:

`https://github.com/microsoft/terminal/tree/master/src/tools/ColorTool`

Further down, `README.md` should have a title **Installing** with a link to the latest `ColorTool` release, with a built `.exe` file and a `schemes` directory. Here is the current one upon writing this:

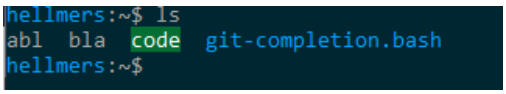
`https://github.com/microsoft/terminal/releases/tag/1904.29002`

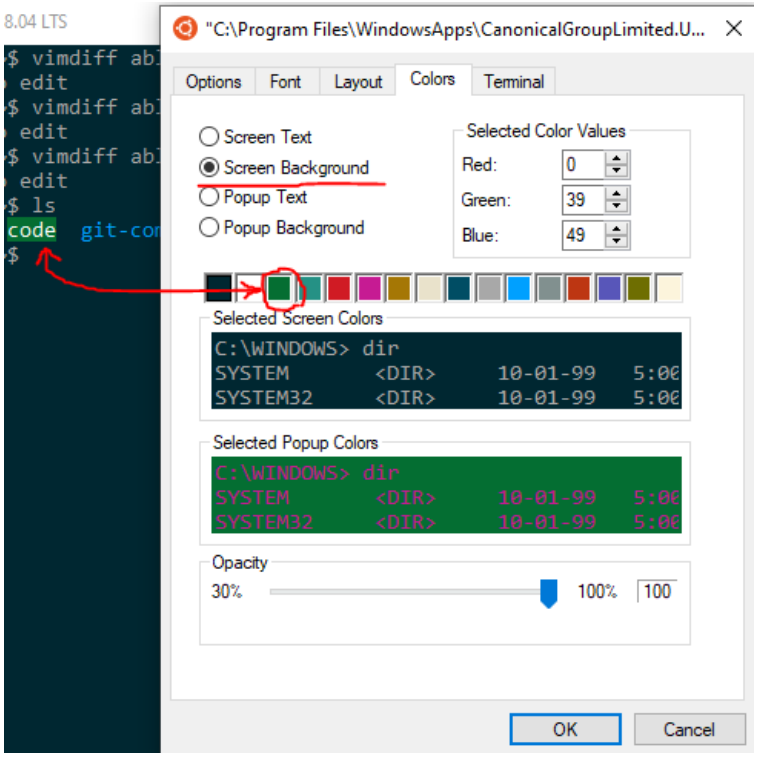
I currently use the `solarized_dark.itermcolors` scheme with some additional manual adjustment in properties, described in section 1.3.3.

1. Download the `zip` file.
2. Open `Command Prompt` in Windows.
3. Write `PATH` or `echo %PATH%` if that doesn't work, to see the different paths.
4. Find a suitable location, such as `C:\Users\Robin.Hellmers\AppData\Local\Microsoft\WindowsApps`, to unzip the files. Unzip them.
5. Open `Command prompt` at the very same directory.
6. Check the `schemes` directory for the names of the different schemes e.g. `campbell.ini`, `OneHalfDark.itermcolors`, ...
7. In the `cmd`, run `ColorTool -b {scheme}` e.g. `ColorTool -b solarized_dark.itermcolors`.
8. Restart the Ubuntu app and the color scheme is applied.

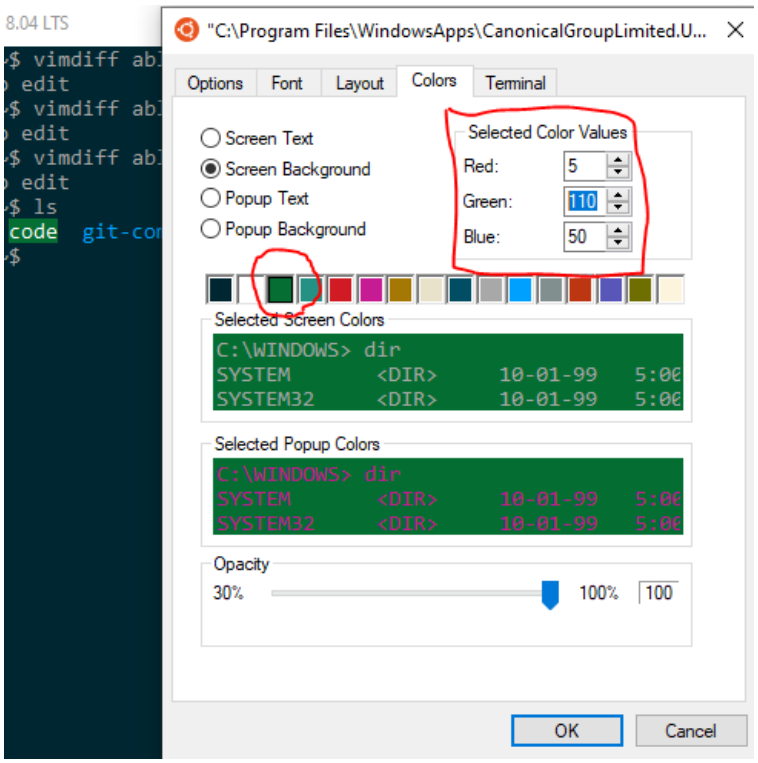
1.3.3 Specific colors with properties tool

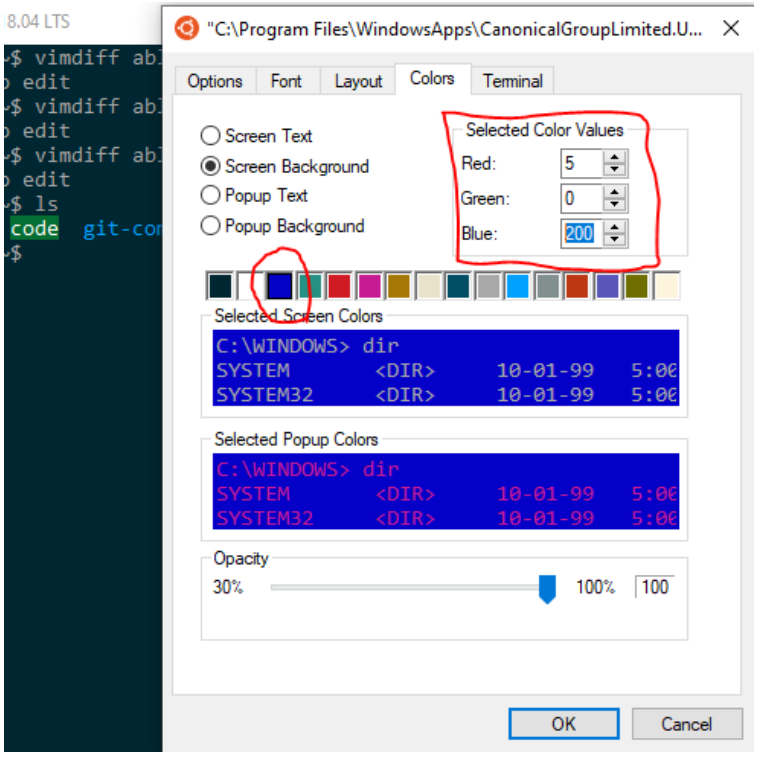
1. Open up the Ubuntu app.
2. Right-click on the top bar which says **Ubuntu 18.04 LTS** besides the icon logo. Press **Properties**. Go to the **Color** tab.
3. It is a weird color tool. When pressing **Screen Text** and **Screen Background**, observe which colors that are highlighted and note it down. These must be selected just before pressing **OK** later on.
   - These two selections are the main colors. The main background and main text. The selected ones, when pressing **OK** becomes the main colors.
   - One have to be careful of changing the colors as it is hard to reset them later on.
4. Lets say tha you want to change this green highlight (Screen Background). Press **Screen Background** and observe which color that is highlighted and thereby is the main color. Press the same green color as the one you want to change.



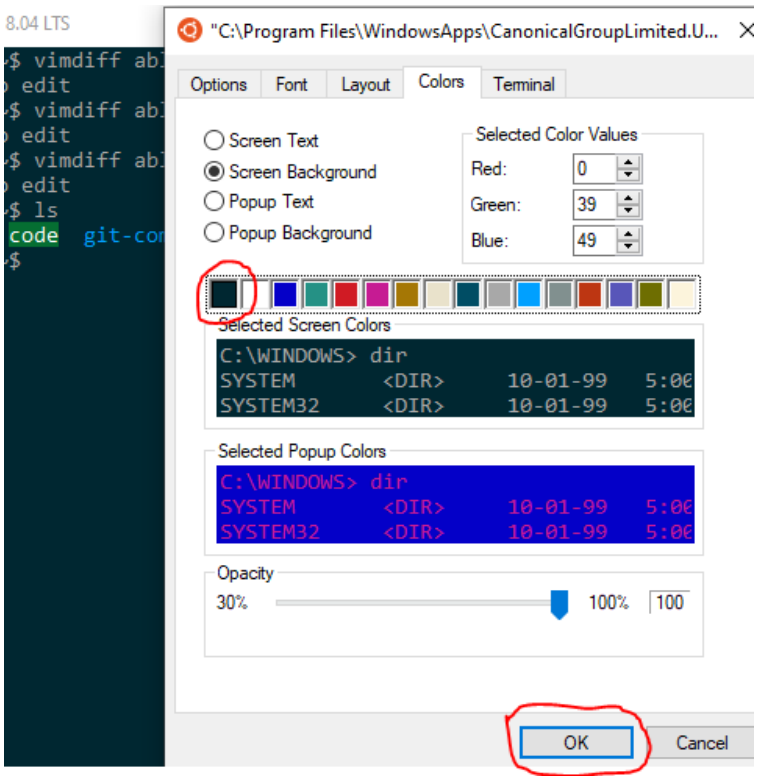


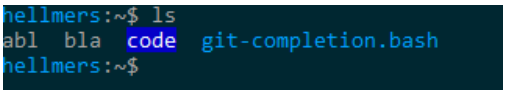
5. As the color you want to change is highlighted, change the RGB values to what you want to have instead.





6. Press the previously highlighted (main color) and then press **OK**. Then the color have been changed.

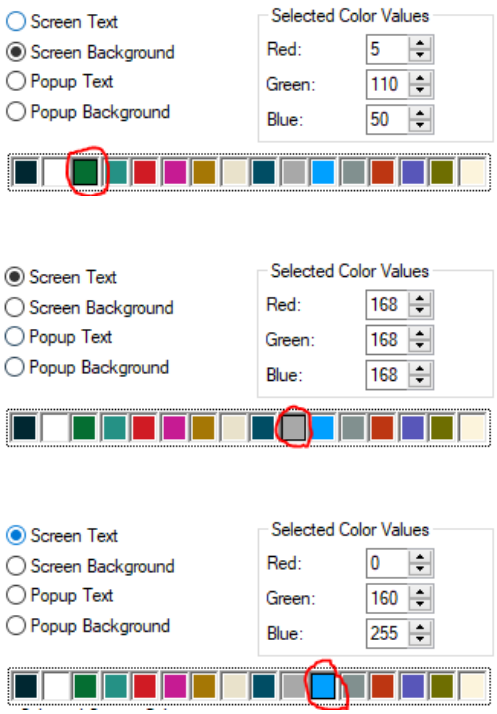




One might want to change a text color for a specific highlight color. This is done by

1. Press **Screen Background**
2. Note which color that is marked. That is the main background color.
3. Select the highlight color which you want to change the text color for
4. Press **Screen Text**
5. Note which color that is marked. That is the main text color.
6. Select the text color you want for that highlight color
7. Press **Screen Background**
8. Select the noted background color from step 2.
    - This will keep your main background color as the last selected one when pressing **OK** is the one becoming the main.
9. Press **Screen Text**
10. Select the noted text color from step 5.
    - This will keep your main text color as the last selected one when pressing **OK** is the one becoming the main.

Here are some of my colors:



1.3.4 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.

Probably will the directories made within Windows have another highlight color and this is because these by default have other reading and writing permissions than the ones made within Ubuntu.

Windows made directories probably have `o+w` permission. Enabling others than the user to write to it. This is called "writable by other". The Ubuntu made ones are probably sticky, which means that they have their sticky bit enabled.

We do not want to change permissions in order to change the colors of the directories. Instead we change the colors of the different permissions and make the two have the same colors.

In `.bashrc` there are some lines trying to access the file `~/.dircolors`. But if you check, there are no such file.

We create it with this command:

```
dircolors -p > ~/.dircolors
```

Then in order to change the colors of the different permissions; find the corresponding keyword and change the numbers. The background color used for the Windows made directories with "writable by other" permission, is set by the variable `OTHER_WRITABLE`. E.g.

```
OTHER_WRITABLE 34;42 # dir that is other-writable (o+w) and not sticky
```

It is probably sufficient to copy this value into the `DIR` variable. From this

```
DIR 01;34 # directory
```

to this

```
DIR 34;42 # directory
```

and it is a good idea to save the default value in a comment

```
DIR 34;42 # default 01;34 directory
```

It could also be a good idea to copy the value into the variables `STICKY_OTHER_VARIABLE` and `STICKY`.

Restart the terminal.

1.3.5 Vimdiff colors

The default colors of `vimdiff` can be really bad because of the translation from 16-bit colors to 256-bit colors.

Code can be found here:

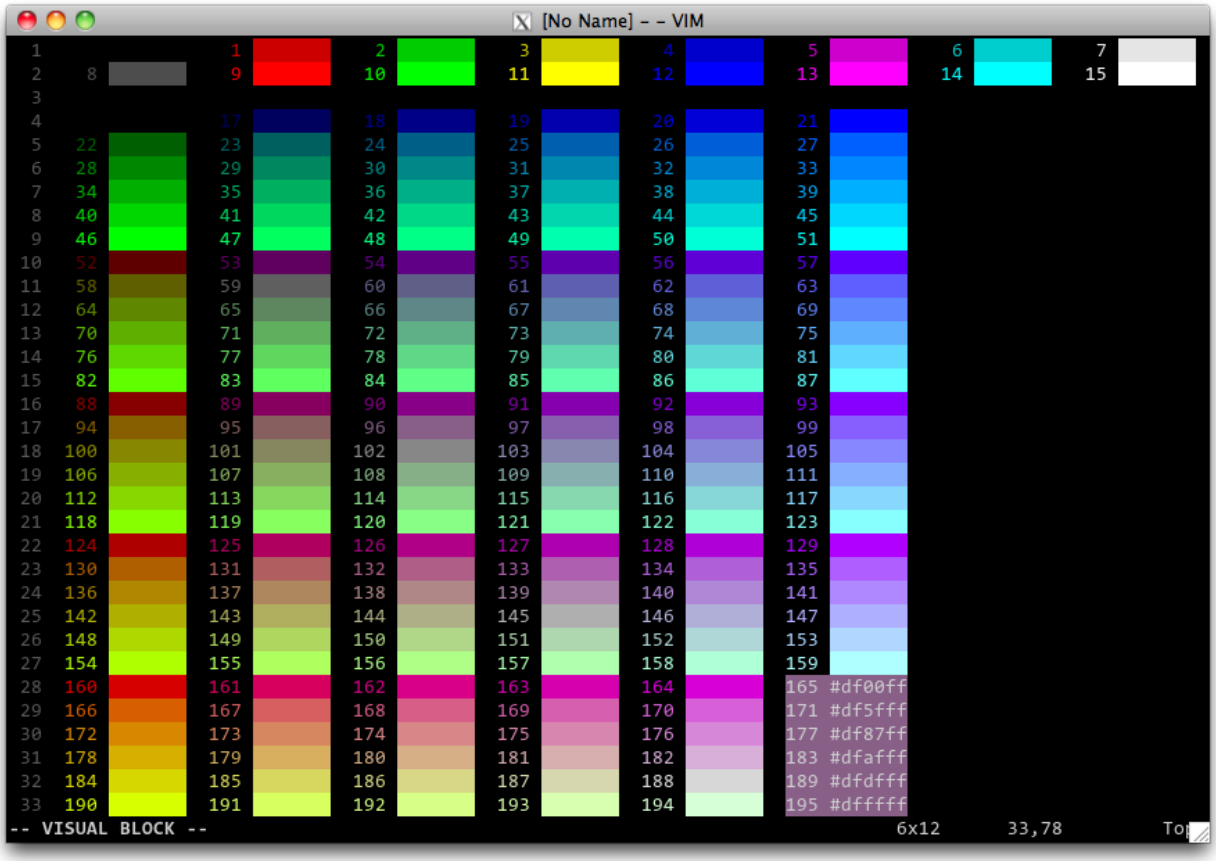
`https://github.com/robinhellmers/computer_setup/`

This can be fixed to something like this instead:

1. Create the `~/.vim/colors/` directory.
2. Create a file `mycolorscheme.vim` in `~/.vim/color/`.
3. Paste this into the file (see Github):

```
highlight DiffAdd    cterm=bold ctermfg=15 ctermbg=22 gui=none guifg=bg guibg=Red
highlight DiffDelete cterm=bold ctermfg=15 ctermbg=88 gui=none guifg=bg guibg=Red
highlight DiffChange cterm=bold ctermfg=15 ctermbg=17 gui=none guifg=bg guibg=Red
highlight DiffText   cterm=bold ctermfg=15 ctermbg=130 gui=none guifg=bg guibg=Red
```

- `ctermfg` = foreground/text color
- `ctermbg` = background/highlight color
- Values given by xterm256 color table. This table might not correspond exactly to what you see on screen. Thereby it is better to print them out manually.



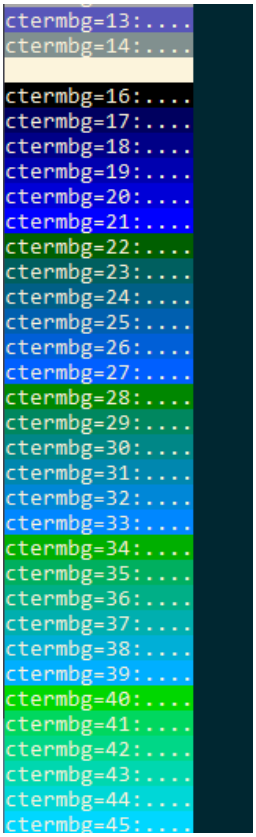
(a) Create a file `color_demo.vim` anywhere.

(b) Paste this into the file (see Github):

```
let num = 255
while num >= 0
    exec 'hi col_'.num.' ctermbg='.num.' ctermfg=white'
    exec 'syn match col_'.num.' "ctermbg='.num.':...." containedIn=ALL'
    call append(0, 'ctermbg='.num.':....')
    let num = num - 1
endwhile
```

(c) Open it up with `vim color_demo.vim` and then use the command `:so color_demo.vim`.

(d) This shows the background colors with corresponding values. Use **Page Up** and **Page down** to go through it.



4. You can continuously edit `~/.vim/colors/mycolorscheme.vim` and see the updates of the colors while still in `vimdiff` by using the command `:colo mycolorscheme`.

5. Now we are going to set this scheme permanently for `vimdiff`. Create a file in your home directory `~/.vimrc`.

6. Paste this into the file (see Github):

```
if &diff
    colorscheme mycolorscheme
endif
```

7. Now the custom color scheme should be applied every time you open `vimdiff`.

1.4 WSL Ubuntu Customization

1.4.1 Step-by-step

1. Download `git-completion.bash` from this Github and copy the code.
2. Create a new file `~/.git-completion.bash` (with dot to make it hidden) and paste the code into it.
3. Use `sudo chmod +x ~/.git-completion.bash` in order to give permission to the user to run it through `.bashrc`.
4. Download `bashrc_shorten_path_git` from this Github and copy the code.
5. Open up `~/.bashrc` and find the code similar to this and replace it with the copied code.

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

1.4.2 Terminal shorten name & path, add git indication

Only this link is needed. The rest below this is the same, Github is used to easily copy the code. Github with `.bashrc` code and `git-completion.bash` forked from official Git source code:
`https://github.com/robinhellmers/computer_setup`

Use `sudo chmod +x ~/git-completion.bash` in order to give permission to the user to run it. Thereby it can run it through `./bashrc`

Instructions for git fetched from here:
`https://git-scm.com/book/id/v2/Appendix-A%3A-Git-in-Other-Environments-Git-in-Bash`

Git source code `git-completion.bash`. Copy the content of the file from official git and add it to your home folder as `git-completion.bash`:
`https://github.com/git/git/blob/master/contrib/completion/git-completion.bash`

Add this above the code that is going to be replaced:

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

Replace the similar code with this:

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

Add this below the code that is going to be replaced:

```
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)
```

Here is all of the above:

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

If something says that permission is denied to the file `path_to_file/git-completion.bash`. Then run `chmod +x path_to_file/git-completion.bash` and restart the Ubuntu app.

2 Virtual Machine Setup

2.1 Installation VirtualBox & Ubuntu 18.04

1. Download .iso file of Ubuntu 18.04.
2. Download and install VirtualBox.
3. Create new virtual machine.
   (a) Version: Ubuntu (64-bit); If not showing 64-bit, enable Virtual Machine in BIOS of host machine.
   (b) Next. Memory 4-5 GB if total 8 GB.
   (c) Next. Select `Create a virtual hard disk now`.
   (d) Create. Select `VDI`.
   (e) Next. Select `Dynamical`.
   (f) Next. 20-40 GB size. More towards 40 GB.
   (g) Create. Wait on creating storage. Done.
4. Settings of virtual machine.
   - → System → Motherboard; Memory still 4-5 GB.
   - → System → Motherboard; Enable I/O APIC
   - → System → Processor; 2 CPUs if total of 4 CPUs.
   - → Display → Screen; Max graphics memory.
5. Start virtual machine. Should ask for `start-up disk` where you `add` the .iso file in `Optical Disk Selector`. If not showing up follow following:
   (a) Go to settings → Storage.
   (b) Mark sub-group to `Controller: IDE`.
   (c) Under Attributes → Optical Drive; Press the circular button to the right.
   (d) Select `Choose/Create a Virtual Optical Disk...`
   (e) Add the .iso file.
   (f) Start virtual machine.
6. Choose to install Ubuntu.
7. Follow the steps and in one of the steps choose `Erase disk and install Ubuntu`. As this is a virtual machine, nothing will be erased on the host computer.

2.2 VirtualBox Extra Setup

2.2.1 Full-screen

1. Start virtual machine.
2. Press `Devices` drop down list in the virtual box window. That is, not inside the virtual machine itself.
3. Press `Insert Guest Additions CD image...`
4. A popup in the virtual machine should show: `... contains software intended to be automatically started. Would you like to run it?` and choose `Run`.
5. Resize the window a little and it will be full-screen.

2.2.2 Shared clipboard

1. In virtual box settings go to `General->Advanced` and select `Bidirectional` for `Shared Clipboard:`
2. Start virtual machine and see if it is working. If not, continue
3. Press `Devices` drop down list in the virtual box window. That is, not inside the virtual machine itself. Press `Insert Guest Additions CD image...`
4. If an error occurs do the following and then redo it
   - Unmount VBoxGuestAdditons by `Devices->Optical Drives->Remove disk from virtual drive`.
5. Reboot the virtual machine.
6. If it is not working, continue
7. Download and install *Extension pack* from virtual box.
8. Reboot the virtual machine.
9. If it is not working, try unmount and mount guest additions again.

2.2.3 Network setup for server and client IPv4 addresses

When starting virtual machine: `Ctrl + Alt + T` for terminal.
Write: `ip addr show`, check wether ip-address is something like `192.11.1.24` and not `10.0.1.1`.
If something with `10.(...)`, then it is a local IPv4 address and not one from the DHCP of the router.

Solution: Turn off virtual machine. Go to → Settings → Network and in `Attached to:` choose `Bridged Adapter` instead of probably NAT.

Start virtual machine and check if IPv4 address have changed to something like `192.(...)`.
If you open up a web-browser and don't get a connection, more settings have to be changed.
This probably depends on the virtual machine giving the router one MAC address and the host computer giving another.

Solution: Turn off virtual machine. Go to → Settings → Network and expand `Advanced`. Remove the MAC address. Then go to the host computer in Windows 10 and open `CMD`. Write: `ipconfig /all` and look for the MAC address of the host machine, probably named something like

```
Physical Address ..........................................  2C-F0-AF-73-2A-6C
```

Input this instead of the old removed MAC address and save. This probably makes you unable to use internet on the host machine instead which one will have to sacrifice.

2.3 VM Ubuntu installations

Everything in section 1.2 should be done.

2.3.1 Visual Studio Code

Install VS Code:

```
sudo snap install -classic code
```

3 Python

3.1 Python3 in Visual Studio Code

`https://stackoverflow.com/questions/50993566/vscode-there-is-no-pip-installer-available-in-the-selected`

4 Extra

4.1 Vim Settings

Create `/.vimrc`.

In order to permanently have numbering in vim/vi/vimdiff, add `:set number` into `.vimrc`.

4.2 Change keyboard layout - MicroSoft Keyboard Layout Creator (MSKLC)

Download MicroSoft Keyboard Layout Creator (MSKLC):
`https://www.microsoft.com/en-us/download/details.aspx?id=102134`

Extract the files and run the installation. If the installation says that `.NET Framework` has to be installed, then follow the instructions below.

1. Press `Windows + R`.Run `appwiz.cpl`.
2. Click `'Turn Windows features on or off'`.
3. Enable `.NET Framework 3.5 (includes .NET 2.0 and 3.0)`.
4. Follow the on-screen instructions and let Windows install the necessary files.
5. Now try to install `MSKLC` again.

Now run `Microsoft Keyboard Layout Creator ...`.

1. Click `File` → `Load Existing Keyboard` and pick the keyboard which the computer have e.g. `Swedish`
2. Now save a backup by clicking `File` → `Save Source File As...`.
3. Now change the name of the current keyboard to avoid saving on the backup after doing the changes. Click `Project` → `Properties` and change the name etc. as before but to another name such as `Custom`.
4. Now, do the changes to the keyboard layout as you which.
   - { mapped to `AltGr + A`
   - [mapped to `AltGr + S`
   -] mapped to `AltGr + D`
   - } mapped to `AltGr + F`
   - \ mapped to `AltGr + G`
5. Run the validation by clicking `Project` → `Validate Layout` and watch for errors. The warnings can usually be ignored.

6. Test write with the layout by clicking `Project` → `Test Keyboard Layout`.
7. Create the necessary files for the final keyboard layout by clicking `Project` → `Build DLL and Setup Package`. When asked about opening the directory where the files were created, click `Yes`. Usually created in `...\Documents`.
8. Run the `setup` file and restart the computer.

4.3 Windows code compare program - Meld

Comparing and inserting code. Great when someone sends you a changed file and might want to integrate it into yours. `https://meldmerge.org/`