Linux Surface:
Setting Up Ubuntu + i3 on a Surface Pro

Brendon Anderson

02 January 2019

Disclaimer

The use of any of the information provided in this document is *at your own risk*. The author of this document is in no way liable for losses or damages that arise in connection to this document. The author makes no guarantees or warranties regarding the accuracy or effectiveness of the methods proposed in this document.

Preface

This document outlines the process I took to set up and customize Ubuntu 18.04 LTS on my Microsoft Surface Pro 2017 (also called the Surface Pro 5). I initially started documenting this process for myself, so that in the case I ever needed to set up Ubuntu on my Surface Pro from a clean slate, I'd have something to reference. I quickly came to realize that there exists an entire community of Surface owners (e.g. this Subreddit) who want to make the switch to GNU/Linux platforms, and therefore others may benefit from this document as well. As this is my first GNU/Linux machine, many of the steps outlined in this document are rather trivial. Therefore, the intended audience of this report consists of those who are new to GNU/Linux operating systems, but perhaps even more experienced users who are first setting up their Surface device on a GNU/Linux platform may find use in this outline.

If you find mistakes, typos, better approaches, or have questions, please feel free to contact me at bganderson@berkeley.edu.

Update: 21 August 2020

Starting later this month, I need a reliable computer that allows me to simultaneously use the front-facing camera and digitally draw with a pen. Ultimately, my Linux Surface setup lacked in these aspects, despite the fun, efficiency, and flexibility it gained in other areas. Therefore, I have decided to wipe my Surface Pro and reinstall a clean and up-to-date version of Windows 10 as my primary operating system. Using the newly released Windows Subsystem for Linux 2 (WSL 2), I have setup an Ubuntu distribution natively within Windows. When working on more terminal-based projects, such as LaTeX documents or Python code, I'm using my Ubuntu OS with tools such as Vim and ranger that are setup similarly as described in this document. For more GUI-esque processes that require sophisticated and reliable hardware integration, such as launching, recording, and screen-sharing in Zoom meetings, and digital drawing/writing with the Surface pen, I'm using the native Windows 10 tools instead of those described in this document. As a result, I no longer use many of the setups that follow, and I won't be updating or expanding this document for the foreseeable future. On the other hand, if my new setup using WSL 2 continues to grow into a sophisticated and powerful amalgam of tools, I may end up writing a new document outlining the corresponding steps taken to build it.

Contents

1 Setting Up Ubuntu

1.1 Ubuntu Installation

Since this is my first time running a machine on a GNU/Linux platform, I chose to use one of the most popular and well documented distributions: Ubuntu. Ubuntu is a derivative of the Debian distribution, and it is offered as a LTS (long-term support) installation, which receives official support and updates from Canonical for 5 years. The installation is carried out by first creating a bootable USB drive according to this article. Once the USB drive is bootable, the Surface Pro can be booted from the USB drive by shutting down the machine, then, while holding the `+` volume button, turning the machine back on. The Surface Pro should enter its UEFI (similar to BIOS found in older machines). Under the `Boot Configuration` tab, swiping left on the `USB Storage` device option will then restart the machine on Ubuntu from the bootable USB drive.

Once the machine finishes booting, you can choose to test Ubuntu without performing any installation, or directly proceed with the installation process. I chose to first test Ubuntu to ensure that the Surface Pro natively had at least some basic functionalities, such as the keyboard and trackpad. Indeed I was able to navigate Ubuntu without any problems. I then chose to connect to the internet using Ubuntu's built-in WiFi modules so that the machine has internet access during the installation. At this point I proceeded with the installation process. By double clicking the icon on the desktop, the installation can be initiated from within the "testing" environment running on the USB drive. The installation process is straight-forward, with only a few decisions necessary. On the `Updates and Software` screen of the installation GUI, I chose the `Minimal installation`, `Downloads updates`, and `Install third-party software` options. On the `Installation Type` screen of the installation GUI, I chose the `Erase disk and install` option, since I wanted to completely remove my Windows installation. If you'd like to install Ubuntu as a secondary operating system and keep your current OS, this is the screen where you inform the installer to do that. The remaining steps of the installation process are self-explanatory.

1.1.1 Updates

After finishing the installation of Ubuntu and rebooting the machine, Ubuntu prompted me to install software updates, to which I accepted. I then opened the terminal and used the following inputs to update any remaining packages, as well as the Ubuntu distribution itself:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get dist-upgrade
```

1.2 Surface Kernel

As of now, Ubuntu does not natively support many of the hardware the Surface Pro has to offer. For example, the touchscreen and pen are not supported. To resolve this issue, GitHub user `jakeday` has developed a custom Linux kernel (read more on kernels here). Furthermore, the GitHub user `qzed` added ACPI support (kernel module) to `jakeday`'s kernel so that Ubuntu can interact with the Surface Pro battery in order to display accurate battery percentage readings. Currently, even these modified kernels do not provide support for some functionalities, such as the Surface Pro built-in cameras. Be sure to read through the documentation before installing any kernel so that you know what features you are gaining access to.

At the time I installed the custom Surface Pro kernel, `jakeday` had not merged `qzed`'s ACPI module into his repository, and therefore I downloaded a pre-compiled kernel pre-released by `qzed` found here. Only the three `.deb` files and the single source code `.zip` file must be downloaded. To begin the installation of the custom kernel, unzip the source code `.zip` file and open a terminal. Navigate inside of the unzipped source code directory and run the `setup.sh` script by typing the following command into the terminal:

```
sudo sh setup.sh
```

The script will prompt you with a different options. Note that, at the time I ran the script, affirmative answers were *only* recognized by typing `yes`, and not by `y`, `Y`, or any other common variation of affirmative inputs. I chose `yes` for all prompts, *except* for the prompt which asks whether to automatically download and install the latest kernel version files. The reason I input `no` for automatically downloading kernel files is as follows: at the time I downloaded the kernel from `qzed`'s repository, `jakeday` had not merged the ACPI support into his repository. Furthermore, the `setup.sh` script still linked to the old `jakeday` kernel files (without ACPI support). I believe that since the time I ran the `setup.sh` script, `jakeday` has merged the ACPI support into his repository and therefore I imagine the latest version of the `setup.sh` script would correctly link to the kernel with ACPI support. I suggest inspecting `setup.sh` to ensure this is true. In any case, I needed to proceed with the kernel installation manually by opening a terminal, navigating to the directory which contains the three `.deb` files (and no other `.deb` files), then running the following command:

```
sudo dpkg -i *.deb
```

After this installation process finishes and rebooting the machine, Ubuntu should now recognize touchscreen inputs, pen inputs (aside from the eraser button), and the GNOME desktop environment should display the battery icon in the bar at the top-right of the screen. If you'd like to have the battery percentage displayed, download the GNOME Tweak Tool by inserting `sudo apt-get install gnome-tweak-tool` into a terminal, then open the Tweak Tool and enable the battery percentage indicator option.

1.3 Trackpad Gestures

One thing I noticed was that Ubuntu did not support advanced trackpad gestures (i.e. three- and four-finger gestures). Luckily, there exists a multitouch gesture recognizer called Fusuma which assigns keybindings or commands to certain gestures. The installation and set-up process is outlined here. As a brief summary, Fusuma can be installed by typing the following into a terminal:

```
sudo gem install fusuma
```

Next, create a configuration file as follows:

```
touch ~/.config/fusuma/config.yml
```

In the Fusuma configuration file you are able to define which commands to perform upon completion of a given gesture. My Fusuma configuration file is as follows:

```
swipe:
  3:
    left:
      command: 'xdotool key alt+Right'
    right:
      command: 'xdotool key alt+Left'
```

```
    up:
      command: ''
    down:
      command: ''
  4:
    left:
      command: 'xdotool key super+Tab'
    right:
      command: 'xdotool key super+Shift+Tab'
    up:
      command: ''
    down:
      command: ''
pinch:
  in:
    command: 'xdotool key ctrl+plus'
  out:
     command: 'xdotool key ctrl+minus'

threshold:
  swipe: 0.4
  pinch: 0.4

interval:
  swipe: 0.8
  pinch: 0.1
```

In this manner, three-finger swipes are useful in changing tabs within Chromium, and four-finger swipes are useful in changing workspaces within i3.

Now, to start recognizing these gestures, a Fusuma process must be running. By default, I was required to run Fusuma using root privilege (i.e. `sudo`). This was problematic, since Fusuma should ideally launch as an automated startup process so that the desired gestures are always recognized. To resolve this problem, I created a file as follows:

```
sudo visudo -f /etc/sudoers.d/fusuma
```

Note that using `visudo` permits editing `sudoer` files in a safe manner (see more here). Inside of the file, I added the following line:

```
ALL ALL= (root) NOPASSWD: /usr/local/bin/fusuma
```

This allows the command `sudo fusuma` to be run as root by all users without password input. Fusuma can now be launched as a startup application by running the command `sudo fusuma` (be sure to include `sudo`).

1.4 Hibernate

Note: I use Vim as my text editor in the steps outlined in this section. If you do not yet have Vim installed, you can substitute the commands `vim` with the default Ubuntu editor `gedit` or `nano`.

At the time of installation, the `jakeday` kernel did not support suspend, and therefore the next best option is to use hibernate when closing the lid of your Surface machine. By default,

my system would not hibernate, even after selecting to replace suspend with hibernate during the custom kernel installation. The steps that follow are those I needed to take in order for hibernate to work properly on my machine. First, type the following into a terminal:

```
sudo vim /etc/systemd/sleep.conf
```

In that file, change `SuspendState=freeze` to `SuspendState=disk`. Next, in order to use hibernate, the system needs either a swap partition or a swap file to suspend the state of the system to disk, i.e. there needs to be memory allocated to the hibernate process. Starting from Ubuntu 18.04, a 2GB swap file is automatically generated in lieu of a partition. According to Ubuntu's website, the swap file must be at least the size of the system's RAM, plus a bit more for overflow. Thus, for an 8GB RAM system, they recommend using an 11GB swap file. By default, the swap file is located as `/swapfile`. To see how much memory is dedicated for swap, type the command `free` into a terminal.

To increase the swap file from 2GB to 11GB, first disable and delete the current swap file:

```
sudo swapoff /swapfile
sudo rm /swapfile
```

Then, create the new swap file as follows:

```
sudo fallocate -l 11GiB /swapfile
```

Next, assign root user read/write permissions to the swap file:

```
sudo chmod 600 /swapfile
```

Then, format the file as a swap file:

```
sudo mkswap /swapfile
```

The new swap file should automatically be enabled at boot. If you must manually enable the swap file, you can do so by typing the following into a terminal:

```
sudo swapon /swapfile
```

To verify that the new swap file is enabled, type `free` into a terminal. If the swap file is disabled, the memory allocated to the swap will be `0`. To find which device the root partition is located on, type the following into a terminal:

```
sudo lsblk -o NAME,FSTYPE,SIZE,MOUNTPOINT,UUID
```

Next, look for the device containing the swap file (the device which has type `ext4` and mount point `/`). Copy the `UUID` value. In my case, this was `eacd7ac6-16e2-4080-b299-c4d924b36a8b`. Now, install the following package:

```
sudo apt-get install uswsusp
```

Next, type the following into a terminal:

```
sudo swap-offset /swapfile
```

In my case, the following was output:

```
resume offset = 17475584
```

Next, edit the `grub` file as follows:

```
sudo vim /etc/default/grub
```

Update the `GRUB_CMDLINE_LINUX_DEFAULT` variable by appending the `UUID` and `resume offset` values found in the previous steps. In my case, the resulting `GRUB_CMDLINE_LINUX_DEFAULT` variable is defined in the `grub` file as follows:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash resume=UUID=eacd7ac6-16e2-4080-
    b299-c4d924b36a8b resume_offset=17475584 resumedelay=15"
```

Note that the `resumedelay` input is an optional delay (in seconds) which is implemented before the machine reads the resume files after hibernate. Now, update the GRUB configuration (which is located at `/boot/grub/grub.cfg` and is overwritten at each kernel update) by running the following:

```
sudo update-grub
```

Now, edit the `resume` file as follows:

```
sudo vim /etc/initramfs-tools/conf.d/resume
```

If the file doesn't already exist, create it by simply replacing `vim` in the above terminal input by `touch`, then begin editing the file. Add the follwing line to the `resume` file, making sure to replace the values for `UUID` and `resume_offset` with your own:

```
RESUME=UUID=eacd7ac6-16e2-4080-b299-c4d924b36a8b resume_offset=17475584 #
    Resume from /swapfile
```

Now, update the `initramfs` files in the kernel image using the following terminal input:

```
sudo update-initramfs -u -k all
```

At this point, closing the lid of the machine or typing `sudo systemctl hibernate` into a terminal successfully put my machine into hibernate. Furthermore, pressing the power button on the top of the Surface resumes the machine in its pre-hibernate state. However, I noticed that after hibernating once, the system would always launch GRUB upon awakening from two or more hibernations, then wait for me to choose to boot Ubuntu. To prevent this, I added the following line to `/etc/default/grub`:

```
GRUB_RECORDFAIL_TIMEOUT=0
```

After updating the GRUB configuration file and kernel images, I noticed that indeed this prevented GRUB from appearing, but plenty of other issues arose after awakening. Therefore, I reverted to the original GRUB configuration. In the end, the following problems still remain:

1. Logging out and back into i3 still kills everything.
2. After hibernating, natural scrolling and tap-to-click seems to turn off.
3. Hibernate and suspend do not work from the LightDM screen (see Section 2.13).

2 Installing Programs

In this section I will describe the process I took to install the programs I use. Clearly, your needs and preferences are different from mine, and therefore your process will inherently vary from the following. In any case, there are a few programs which have slight tricks to get running smoothly, so if documenting my process can help anyone then it is worth writing out.

2.1 Internet Browser

I chose to use Chromium, which is Google's open source backend to Chrome. To install, type the following into a terminal:

```
sudo apt-get install chromium-browser
```

After using Chromium for a while, you may experience pesty pop-up windows requesting you to unlock your keyring. To disable this, create a configuration file for Chromium flags by typing the following into a terminal:

```
touch ~/.config/chromium-flags.conf
```

Now, begin editing the file (e.g. by typing `gedit ~/.config/chromium-flags.conf`) and add the following line:

```
--password-store=basic
```

Save the file and hopefully Chromium will stop pestering you with the keyring popups (although sometimes this fix seemed to fail).

If you'd like to completely remove Firefox (which is installed by default) from your system, type the following into a terminal:

```
sudo apt-get purge firefox
sudo apt-get purge firefox-locale-en
sudo rm -rf /etc/firefox
```

Note that using `purge` instead of `remove` also removes the program's configuration files.

2.2 Music Player

I chose to use the fast, terminal-based music player called cmus. To install cmus, type the following into a terminal:

```
sudo apt-get install cmus
```

You can navigate through the different views in cmus (library, playlists, settings, etc.) by pressing numbers between `1` and `7`. To begin adding music to your library, press `5` and navigate to your music files of interest. With the file (or folder) of interest highlighted, press the `a` key to add the music to your library, whose default view can be found by pressing `1`.

In order for the system's application launcher (such as Rofi or dmenu) to recognize cmus, the program needs a desktop entry. If the file does not already exist, create a `.desktop` file for cmus using the following terminal input:

```
sudo touch /usr/share/applications/cmus.desktop
```

Once the file is created, open it and enter the following lines:

```
[Desktop Entry]
Encoding=UTF-8
Version=1.0
Type=Application
Display=true
Exec=cmus %f
Terminal=true
Name=cmus
Comment=Music player cmus
```

Now, cmus should be recognized by Rofi (so long as Rofi is configured to show all applications with desktop entries, i.e. using `rofi -modi drun -show drun`).

2.3 Email Client

Mutt is a lightweight terminal-based email client. To install Mutt, input the following into a terminal:

```
sudo apt-get install mutt
```

To set up Mutt, begin with the following:

```
mkdir -p ~/.mutt/cache/headers
mkdir ~/.mutt/cache/bodies
touch ~/.mutt/certificates
touch ~/.mutt/muttrc
```

Now you must input your email service information into the `muttrc` configuration file. There is a plethora of sample configuration files online which you can use as a guide. I found that setting up a Microsoft email service (`live.com` to be exact) worked well when following this as a guide.

In order to automatically have Ubuntu open Mutt when clicking on an email address (a `mailto:` hyperlink), I edited `/usr/share/applications/mutt.desktop` and updated to the following lines:

```
Exec=i3-sensible-terminal -e mutt %u
NoDisplay=false
Terminal=false
```

I was able to verify that the hyperlink behavior worked as expected when clicking email addresses in both Chromium and MuPDF.

2.4 Notepad/Drawing

I chose to use Xournal for a pen-input notepad, or for making sketches. This seems to be a popular OneNote alternative for Linux users, especially since it supports pressure sensitivity. To install Xournal, type the following into a terminal:

```
sudo apt-get install xournal
```

Personally, I do not like the lined paper set by default. To change this to plain (blank) paper, open Xournal, click the `Page` menu button in the top bar, navigate down to the `Paper Style` submenu, and select the `plain` option. To set this as the default paper style, click the `Set As Default` option at the bottom of the `Page` menu.

Now, to get the most out of the Surface Pen in Xournal, navigate to the `Options` menu in the top bar, then to the `Pen and Touch` submenu. I enabled the following options: `Eraser Tip`,

`Pressure Sensitivity`, `Touchscreen as Hand Tool`, and `Pen disables Touch`. I also chose the `Select Region` option within the `Button 2 Mapping` submenu, which allows you to use the small button near the writing tip of the Surface Pen to perform selection within Xournal. Be sure to save these preferences by selecting the `Save Preferences` option within the `Options` menu.

At this point, Xournal should support variable-pressure writing with the Surface Pen, as well as erasing with the Surface Pen's eraser tip. I've noticed that on occasion, Xournal does not recognize writing inputs from the Surface Pen on the first launch of the program, and it does not recognize eraser inputs on the second launch of the program. After launching Xournal for the third time, everything functions as intended. Supposedly, installing the package `libcanberra-gtk-module` should resolve this issue, but it has not worked for me. This is a problem I will be looking into in the near future.

2.5 Document Viewer

I chose to install MuPDF as a replacement for Ubuntu's default Evince. I initially tried Zathura, but found that it's lack of support for HiDPI displays resulted in a blurry PDF viewing experience. MuPDF gave crystal-clear viewing and therefore provides a lightweight, minimalist alternative to Evince (I'd like my PDF viewer to be as light as possible to ensure fast previews when compiling LaTeX documents). Install MuPDF by inputting the following into a terminal:

```
sudo apt-get install mupdf
```

2.6 Terminal Emulator

URxvt is a very popular terminal emulator, since it's lightweight, fast, and permits a high degree of customization. I therefore chose to use URxvt as a replacement for the default GNOME Terminal. Install URxvt by inputting the following into a terminal:

```
sudo apt-get install rxvt-unicode
```

When you first start URxvt, it will likely look unpleasant. However, URxvt can be customized by creating the an `.Xresources` file as follows:

```
touch ~/.Xresources
```

The customization process will be discussed further in Section 4.

2.7 Text Editor

I chose to install Vim, which is a very popular, highly configurable terminal-based text editor. Another popular editor is Emacs, which I have yet to try. Rather humorously, there exists an odd tension between Vim and Emacs users, and the arguments for using one editor versus the other has been a hot debate for some time now (see more here).

I decided to install the `vim-gtk` package, rather than the standard `vim` package, since the `vim-gtk` package comes with the `clipboard` feature, allowing you to copy and paste between Vim and the system clipboard. You can verify that your installation of Vim has access to the system clipboard by typing the following into a terminal:

```
vim --version | grep clipboard
```

If your installation of Vim has access to the system clipboard, the terminal output should display `+clipboard`. With this feature, selecting text in Vim, then typing `"+y` will yank (copy) the selection

to the system clipboard, which you can then paste in other programs (such as Chromium, etc.) using whatever keybinding that program may call for (commonly `<CTRL>+v`).

Now, to install the `vim-gtk` package, type the following into a terminal:

```
sudo apt-get purge vim-tiny
sudo apt-get install vim-gtk
```

Next, I decided to perform some quick configurations to make Vim a bit more user-friendly. To do this, first create a Vim configuration file (if one does not already exist):

```
touch ~/.vimrc
```

Now, I like to see which line number I am on when editing documents. You can enable this feature in Vim by adding the following line to the `.vimrc` file:

```
set number
```

Furthermore, I find that scrolling through a text document using a mouse or trackpad is sometimes easier or quicker than using the editor's navigation commands. To enable mouse (and trackpad) scrolling, add the following line to the `.vimrc` file:

```
set mouse=a
```

To learn how to use Vim, a good starting point is the built-in tutorial, which can be accessed by typing the following into a terminal:

```
vimtutor
```

2.8 LaTeX

I typeset my documents using LaTeX. To install LaTeX, type the following into a terminal:

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

The package `latexmk` provides handy compilation tools such as PDF previewing and automated multi-compilation to ensure the update of references and bibliography changes. To install `latexmk`, type the following into a terminal:

```
sudo apt-get install latexmk
```

To set the default program that `latexmk` calls to open a PDF preview to MuPDF, create a configuration file as follows:

```
touch ~/.latexmkrc
```

Next, add the following to the `.latexmkrc` file:

```
$pdf_previewer = 'mupdf';
```

Setting up streamlined compilation in Vim will be discussed in Section 4.

2.9 File Manager

I chose to use ranger as my file manager. It is a lightweight, terminal based, Vim-inspired file manager with plenty of keybinding options. To install ranger, type the following into a terminal (note the \ character used to break the input into two lines):

```
sudo apt-get install ranger caca-utils highlight atool \
w3m poppler-utils mediainfo
```

To set up ranger's configuration files, open ranger by typing `ranger` into a terminal. After ranger loads, quit by pressing `Q`. Now, type the following into a terminal:

```
ranger --copy-config=all
```

The default ranger configuration should now be finished.

2.10 Version Control

Version control systems are useful in development processes to keep track of changes in documents and software (see more here). Two of the most popular version control systems are Git and Apache Subversion. Apache Subversion utilizes centralized repositories, whereas Git implements a decentralized approach. In this fashion, Git allows you to commit changes to a local repository without the need for access to the centralized repository (i.e. internet access).

I chose to install Git. You can do this by typing the following into a terminal:

```
sudo apt-get install git
```

Note that in order to start making commits to a repository, you must configure Git as follows:

```
git config --global user.name "NAME"
git config --global user.email "EMAIL"
```

This information is used to associate commits to a repository with an identity. The values you use for `NAME` and `EMAIL` may depend on which repository hosting service you use, for example GitHub or Bitbucket, and therefore you should consult their documentation for more information.

2.11 Google Drive

The package titled `google-drive-ocamlfuse` is a nifty tool to mount your Google Drive to your GNU/Linux system. `google-drive-ocamlfuse` acts like a USB drive in the sense that none of the files are stored locally on your machine's hard drive. Instead, the files in your Google Drive are accessible (in both a read and write sense) over the internet. To install `google-drive-ocamlfuse`, type the following into a terminal:

```
sudo add-apt-repository ppa:alessandro-strada/ppa
sudo apt-get update
sudo apt-get install google-drive-ocamlfuse
```

To set up `google-drive-ocamlfuse`, start by typing `google-drive-ocamlfuse` into a terminal. This command will generate the program's configuration file, and open an internet browser window which prompts you to input your Google login credentials. Next, generate a folder (locally on your machine) in which you'd like to mount your Google Drive. For example, input the following into a terminal:

```
mkdir ~/google-drive
```

Next, mount your Google Drive into this folder by typing the following mounting command into a terminal:

```
google-drive-ocamlfuse ~/google-drive
```

You may now access your Google Drive files by navigating to `~/google-drive` either using your file manager or in a terminal emulator. Notice that when you delete Google Drive files from your file manager or terminal emulator (for instance, by using the `rm` command), they are moved to

the Trash folder in your Google Drive, which can be found locally at `~/google-drive/.Trash`. If you'd like to automatically have access to your Google Drive at each boot, you can simply add the `google-drive-ocamlfuse` mounting command to your startup applications (for example in Startup Applications when using GNOME, or in your i3 configuration file when using i3 windows manager).

You can now place music files in your Google Drive and play them in your music player (such as cmus) without having to store the files locally by simply specifying the correct local path to your music player. I've found this technique to work very well with cmus, although the process of initially adding music to my library took much longer than if the files were actually stored on my machine. However, once the library was cached, cmus has been able to load and start playing the music very quickly. You can also now use Google Drive to host your Git repositories. Here is an instructional video describing the process.

2.11.1 Syncing a Second Machine

In my specific case, I have a second machine (Windows-based) which is dedicated for more resource intensive work (it is a mobile workstation). I typically leave this machine at home, but bring my Surface Pro around with me during my day-to-day tasks. In the past, I've found myself working on projects using my Surface Pro while out from home, then returning to my mobile workstation at home and using that machine to finish things up or run finalized simulations/code due to its superior capabilities in the realm of processing power. To do this, I used to use a USB drive to exchange files between the two machines. Obviously this is less than ideal, and therefore I would like to be able to automatically sync files between the two machines.

My solution to this syncing dilemma is Google's Backup and Sync utility, which can be downloaded here. This utility allows you to do two things: 1) you can sync your computer to your Google Drive, and 2) you can sync your Google Drive to your computer. The difference between these two functionalities is minimal, but in order to get syncing between two machines to work, the second option works nicely. The reason the first option yields a headache is due to the fact that syncing your computer to your Google Drive ends up placing your computer's files into a section of Google Drive titled `Computers`, which is separate from the standard `My Drive` section. On the other hand, using `google-drive-ocamlfuse` to mount your Google Drive to your GNU/Linux system automatically mounts the `My Drive` section, *not* the `Computers` section. In summary, I am using the second functionality in Backup and Sync in order to maintain syncing between the files stored on my Windows machine, my Google Drive, and the Google Drive mount on my Surface Pro. This leaves me with local copies of the files on my Windows machine (which I can edit offline), as well as synced copies of the files in the cloud. There are *not* local copies of these files on my Surface Pro, unless I choose to copy a file from the Google Drive mount onto my local disk. This setup allows me to work efficiently between the two machines without taking up disk space on my little Surface Pro.

Now, setting up Backup and Sync in this configuration is simple. Download Backup and Sync and run the installer. Once it finishes, it will ask you to connect to your Google account. After signing in and giving Backup and Sync permission to access and edit the files on your Google Drive, there will be two set-up screens. The first screen, titled `My Laptop` will ask which folders (locally stored on your machine) you would like to back up to Google Drive. By choosing any of these folders, they will be synced to the `Computers` section of Google Drive. I left all folders in this screen unchecked. On the following screen, titled `Google Drive`, you are asked whether you would like to sync `My Drive` to your computer. I selected yes, then created a folder (locally stored on my machine) where I would like my Google Drive to sync to (this is essentially a mount point, like the

one used to set up `google-drive-ocamlfuse`). I then selected the folders (located in my Google Drive on the cloud) which I wanted to be stored and synced locally on my Windows machine. The initial process of syncing took quite a very long time (a few hours), which may have been expected since I was syncing approximately 250GB of data.

2.12 Matlab

I installed Matlab by following the Linux installation instructions on the MathWorks website. I then created a symbolic link to open Matlab from a terminal window:

```
cd /usr/local/bin
sudo ln -s /usr/local/MATLAB/R2018b/bin/matlab
```

Next, I adjusted Matlab to work with the HiDPI display. To do this, type the following in the Matlab command window:

```
s.matlab.desktop.DisplayScaleFactor.PersonalValue = 2
```

I then updated the font sizes accordingly in the Matlab preferences. Finally, I adjusted the Matlab shortcuts to my preferences.

2.13 Display Manager

By default, Ubuntu comes with the GNOME Display Manager (GDM). I chose to use LightDM instead, which is a popular lightweight and customizable display manager. To install LightDM, type the following into a terminal:

```
sudo apt-get install lightdm
```

The installation process should prompt you whether you'd like to use LightDM as the default display manager, to which I answered affirmatively. Upon reboot, LightDM's greeter should prompt you for your login credentials.

2.14 Uninstalling Programs

To reduce some of the clutter that comes pre-installed with Ubuntu, I uninstalled the default text editors (gedit and nano), document viewer (Evince), terminal emulator (GNOME Terminal), and file manager (Nautilus). This was done using the following terminal inputs:

```
sudo apt-get purge nano
sudo rm -rf ~/.local/share/nano/

sudo apt-get purge gedit
sudo apt-get purge gedit-common

sudo apt-get purge evince
sudo apt-get purge evince-common
sudo rm -rf /usr/lib/x86_64-linux-gnu/evince

sudo apt-get purge gnome-terminal
sudo apt-get purge gnome-terminal-data

sudo apt-get purge nautilus
```

```
sudo rm -rf /usr/lib/x86_64-linux-gnu/nautilus/
sudo rm -rf /usr/bin/nautilus-sendto
sudo rm -rf /usr/share/man/man1/nautilus-sendto.1.gz
sudo rm -rf ~/.local/share/nautilus/
sudo rm -rf ~/.config/nautilus/
```

Note that all of the packages were removed in an independent fashion (they had no dependencies), *except* for `nautilus`, which automatically removed `ubuntu-desktop`.

3 Setting Up i3

3.1 i3 Window Manager

i3 is a popular tiling window manager which permits a high degree of customization. It can be used to essentially replace the standard GNOME desktop environment which comes with Ubuntu 18.04 by default. This means that at the login screen (display manager greeter), you are able to choose to log in to either GNOME or i3. More about i3 can be read here. Furthermore, this video and its two follow-ups also provide a great introduction to the i3 window manager.

To install i3, type the following into a terminal:

```
sudo apt-get install i3
```

The installation will ask you which key you'd like to use as the `Mod` key (I chose to use the Windows key) which is used to navigate and operate i3. You will also be asked if you'd like for i3 to generate a configuration file, to which I accepted.

To ensure i3 is set as the default session manager, type the following into a terminal:

```
sudo update-alternatives --install /usr/bin/x-session-manager x-session-
    manager /usr/bin/i3 60
```

The number `60` at the end sets the priority level higher than the default value of `50` associated with the `gnome-session` session manager. Furthermore, to ensure that i3 is the default window manager, type the following into a terminal, and select choose the i3 option:

```
sudo update-alternatives --config x-window-manager
```

3.2 Scaling Issues

After installing i3, restarting the machine, and logging into i3 (which can be accomplished by choosing the `i3` option from the gear-shaped button on the GNOME display manager greeter), it was immediately clear that there were scaling problems. All of the text was too small, both natively in i3 (such as in the i3-bar at the bottom of the screen), and in third-party programs (such as Chromium). This problem appears to be related to issues with HiDPI display support. After trying a few different techniques to resolve the issue, I found that the following approach worked best. Start by creating a `.Xresources` file (if one does not already exist):

```
touch ~/.Xresources
```

Next, add the following lines to the `.Xresources` file:

```
Xft.dpi: 180
Xft.autohint: 0
Xft.lcdfilter:  lcddefault
Xft.hintstyle:  hintfull
Xft.hinting: 1
Xft.antialias: 1
Xft.rgba: rgb
```

Restart i3, and hopefully everything will display with better scaling.

3.3 Trackpad Issues

The next problem with i3 that I encountered was the inability to tap-to-click using the trackpad, reversed scrolling (at least, opposite of what I am accustomed to), and perhaps most importantly, the inability to right-click with the trackpad. This article outlined the steps necessary to resolve these issues. As a brief summary of the steps, type `xinput list | grep Touchpad` into a terminal and ensure that the trackpad device name is indeed `Microsoft Surface Type Cover Touchpad`. After doing so, add the following three lines to the i3 configuration file, usually located at `~/.config/i3/config`:

```
exec --no-startup-id xinput set-prop "Microsoft Surface Type Cover Touchpad
    " "libinput Tapping Enabled" 1
exec --no-startup-id xinput set-prop "Microsoft Surface Type Cover Touchpad
    " "libinput Natural Scrolling Enabled" 1
exec --no-startup-id xinput set-prop "Microsoft Surface Type Cover Touchpad
    " "libinput Click Method Enabled" 0 1
```

By adding these lines to the the i3 configuration file, the trackpad functionalities will be initiated each time i3 is started from the display manager.

3.4 Audio Controls

To get the volume buttons on the keyboard to work in controlling the audio level of the device, install the PulseAudio Controls package `pactl`. Then, add the following three lines to the i3 configuration file:

```
bindsym XF86AudioRaiseVolume exec --no-startup-id pactl set-sink-volume 0
    +5%
bindsym XF86AudioLowerVolume exec --no-startup-id pactl set-sink-volume 0
    -5%
bindsym XF86AudioMute exec --no-startup-id pactl set-sink-mute 0 toggle
```

3.5 Backlight Issues

I found that the backlight control keys used to vary the screen brightness were not initially working in i3. Apparently this is a common problem that arises when running `xbacklight` in i3. To regain control over the backlight, a program called Light can be installed.

Start by going to the `Releases` tab of the Light program's GitHub page, found here. Download the latest `.deb` file. Open a terminal and navigate into the directory containing the `.deb` file. Ensure that this is the only `.deb` file in the directory, then install the package by typing the following:

```
sudo dpkg -i *.deb
```

Now that the package is installed, the backlight should be able to be controlled via `light`. However, by default I was required to call `light` with root (`sudo`) privileges. To remedy this problem and permit backlight adjustment via keyboard input on the Surface Type Cover, I took the same approach as that outlined in Section 1.3, i.e. type the following into a terminal to start editing a `sudoer` file:

```
sudo visudo -f /etc/sudoers.d/light
```

In the file, add the following line:

```
ALL ALL = (root) NOPASSWD: /usr/bin/light
```

The `light` command should now be able to be called with root privileges without the need for password input. To call `light` upon keyboard presses, add the following two lines to the i3 configuration file found at `~/.config/i3/config`:

```
bindsym XF86MonBrightnessUp exec sudo light -A 10
bindsym XF86MonBrightnessDown exec sudo light -U 10
```

3.6 Screen Tearing Issues

Another problem I came across while using i3 was screen tearing. This issue manifested itself as a large diagonal cut in my screen while scrolling fast. It was most noticeable in Chromium. To fix this problem, I started by creating a configuration file for the Intel graphics as follows:

```
touch /etc/X11/xorg.conf.d/20-intel.conf
```

I then enabled the `TearFree` option in the driver by editing this configuration file. This is the resulting `20-intel.conf` file:

```
Section "Device"
   Identifier  "Intel Graphics"
   Driver      "intel"
   Option      "TripleBuffer" "true"
   Option      "TearFree"     "true"
   Option      "DRI"          "true"
   Option      "AccelMethod"  "sna"
EndSection
```

Further smoothing of the display was achieved by installing the Compton compositor, which is obtained by typing the following into a terminal:

```
sudo apt-get install compton
```

Next, a Compton configuration file is to be generated:

```
touch  ~/.config/compton.conf
```

I found that by adding the following line to the Compton configuration file, my scrolling and video playback appeared smoother:

```
unredir-if-possible = true;
```

To start the Compton compositor upon launching i3, add the following to the i3 configuration file:

```
exec_always --no-startup-id compton --config ~/.config/compton.conf -f -b
```

Running Compton also allows for transparency in the colors of Rofi and URxvt. For more information on Compton, see the GitHub page, or this article.

3.7 Drive Automounting

3.7.1 USB Mounting

By default, i3 did not mount USB drives automatically upon insertion into the Surface Pro. To enable this feature, type the following into a terminal:

```
sudo apt-get install udisks2 udiskie
```

With the `udiskie` package now installed, the automated detection and mounting of USB drives is possible, and can be enabled at startup, for example by placing the following lines in you i3 configuration file (`~/.config/i3/config`):

```
exec --no-startup-id udiskie -aNT2 --no-appindicator
bindsym $mod+u exec udiskie-umount -a
```

Note that the second line defines an i3 keybinding which unmounts the USB drive.

3.7.2 Google Drive Mounting

As outlined in Section 2.11, it is possible to use the `google-drive-ocamlfuse` tool to mount your Google Drive in a similar fashion to that of a USB drive in order to have local control over your remote files. In order to automatically mount Google Drive to the folder `~/google-drive` when starting i3, simply add the following line to the i3 configuration file found at `~/.config/i3/config`:

```
exec --no-startup-id google-drive-ocamlfuse "/home/username/google-drive"
```

Clearly, you should change `/home/username` to your own user's home directory which contains the folder `google-drive`.

3.8 Customization Packages

In order to make the most of i3, I installed the following packages, which can be used later on to customize the style, feel, and performance of i3 and the machine in general.

3.8.1 Polybar

Polybar is a simple and easily customizable status bar. To install Polybar on Ubuntu, see the GitHub page. Be sure to install all dependencies, and if you can't `apt-get` Polybar, then try installing it from the source provided on the GitHub page. After installing Polybar, be sure to comment out the lines containing `bar{*}` in the i3 configuration file, as these lines are used to load the default status bar i3bar.

You can customize Polybar by editing `~/.config/polybar/config`. In particular, I adjusted which data and modules are shown with the following lines:

```
modules-left = i3
modules-center =
modules-right = cpu memory wlan volume xbacklight battery date
powermenu
```

Under `[module/i3]`, I used the following:

```
type = internal/i3

ws-icon-0 = 1 terminal;1 terminal
ws-icon-1 = 2 ranger;2 ranger
ws-icon-2 = 3 browser;3 browser
ws-icon-3 = 4 other;4 other
ws-icon-4 = 5;5
ws-icon-5 = 6;6
```

```
ws-icon-6 = 7;7
ws-icon-7 = 8;8
ws-icon-8 = 9;9
ws-icon-9 = 10 music;10 music
```

and then customized the theme and colors as desired. Under `[module/xbacklight]`, I used the following:

```
type = internal/xbacklight

format = <ramp><label>
label = %percentage%
```

and then customized the `ramp` values to contain symbols representing backlight level. I use the following under `[module/backlight-acpi]`:

```
inherit = module/xbacklight
type = internal/backlight
card = intel_backlight
```

which connects the status bar backlight level to the actual backlight level of the device. For the battery indicator, I used the following under `[module/battery]`:

```
type = internal/battery
battery = BAT1
adapter = AC
full-at = 100
```

which tells the status bar which battery to connect to. I then adjusted the battery level logo, ramps, and animation to my desired customization. The other Polybar features are customized in a similar manner, just look up the options online.

3.8.2 Rofi

Rofi is an application launcher used to replace the standard `dmenu` launcher. Rofi is highly customizable both in terms of aesthetics as well as performance. To install Rofi, type the following into a terminal:

```
sudo apt-get install rofi
```

To bind Rofi to a keybinding (`dmenu` is bound to `mod+d` in i3 by default), I replaced the `dmenu` keybinding in `~/.config/i3/config` with the following:

```
bindsym $mod+d exec rofi -modi drun -show drun
```

I specifically used the `drun` option in order for terminal-based applications (like Mutt or ranger) to be able to launch from Rofi, so long as they have a suitable `.desktop` file (see Section 4.2).

3.8.3 betterlockscreen

Note: It took me a few tries to figure out how to build and install this package and its dependencies before I actually got it to work. I didn't document each successful step as I went, so the following outline is what I remember doing, and what I *think* ended up getting `betterlockscreen` to work. You may need to play around with the installation process or do some more research in order to get your setup to work.

The default lockscreen in i3 is ugly. It is called by `i3lock` and yields a white screen, where you then must type your credentials in order to log back in. While typing, an enormous circle flashes with colors in the middle of the screen.

A much sleeker lockscreen option is offered by `betterlockscreen`, found here. To install `betterlockscreen`, I first had to install two packages as follows:

```
sudo apt-get install autoconf
sudo apt-get install checkinstall
```

I then had to update to the newest version of `i3lock`, which can be found here. I recall that, for some reason, my version of i3 did *not* come with the latest version of `i3lock` installed, and that this caused failures when trying to install `betterlockscreen`. To install the latest version of `i3lock`, I started by downloading the latest `.zip` file from the `Releases` page. I then unzipped the file, and changed directories into the new folder in which the files were unzipped. I then built the package according to the instructions on `betterlockscreen`'s GitHub page:

```
autoreconf --force --install

rm -rf build/
mkdir -p build && cd build/

../configure \
  --prefix=/usr \
  --sysconfdir=/etc \
  --disable-sanitizers

make
```

I then typed the following into a terminal in order to actually install the newly compiled package:

```
checkinstall
```

I then found this very useful script (for Debian/Ubuntu systems). When this script is run, it automatically downloads and installs both `betterlockscreen` and its dependencies (the ones not discussed above). To run the script, download the `.sh` file from the GitHub page. Ensure the script is executable by changing directories into the folder containing the script, and typing the following into a terminal:

```
chmod +x betterlockscreen.sh
```

Once the script is executable, run it by typing `./betterlockscreen` into a terminal. Once installed, type `betterlockscreen --help` into a terminal to get help on how to use `betterlockscreen`.

The next step I took was to create a keybinding for calling `betterlockscreen`. To do this, I put the following line in my `~/.config/i3/config` configuration file:

```
bindsym $mod+shift+x exec betterlockscreen -l blur
```

4 Configuring and Customizing

4.1 Default User Directories

To change the default user directories, open the following file and change the directory names:

```
~/.config/user-dirs.dirs
```

Personally, I just renamed the directories to have all lowercase lettering, and removed the `Public` and `Templates` directories.

4.2 Default Programs

To set the system's default programs, open the following file and change the program associated with each file type to those which you'd like:

```
/usr/share/applications/defaults.list
```

This file tells your system which program handles a given file type. Note that these can also be set by the `xdg-mime` command. The changes I made to `defaults.list` include the replacement of `evince.desktop` with `mupdf.desktop`, `rhythmbox.desktop` with `cmus.desktop`, `gedit.desktop` with `vim.desktop`, `thunderbird.desktop` with `mutt.desktop`, and lastly `firefox.desktop` with `chromium-browser.desktop`.

Furthermore, you can change the default program called by other programs or commands by using the `update-alternatives` command. For example, to change the default text editor opened when the command `editor` is called, type the following:

```
sudo update-alternatives --config editor
```

For a list of the commands you can update, type the following:

```
update-alternatives --get-selections
```

The only manually updated alternative I set was `x-terminal-emulator` to `urxvt`. I also installed vim as an alternative for `pico` and `gnome-text-editor`, just to ensure vim was default for everything text-related, and set its priority high enough to ensure it as default:

```
sudo update-alternatives --install /usr/bin/pico pico /usr/bin/vim.gtk 60
sudo update-alternatives --install /usr/bin/gnome-text-editor gnome-text-
    editor /usr/bin/vim.gtk 60
```

4.3 Clock Synchronization

For reasons unclear to me, I found my system's clock to be miscalibrated at some point during this set-up process. The clock displayed the wrong time until an internet connection was formed, at which point the clock would adjust itself correctly. I found that entering the following into a terminal solved this problem:

```
sudo timedatectl set-ntp true
```

To verify that the synchronization was successfull, type the following:

```
timedatectl status
```

4.4 Terminal Customization

4.4.1 Terminal Prompt

By default, when a terminal emulator is opened it prompts the user for input by displaying the following:

```
user@host:~$
```

Now, if your hostname (`host` in the example above) is long, then you will find your terminal inputs may start to look cluttered. I decided to shorten the prompt to only show my username, which looks as follows:

```
user:~$
```

To do this, open the `.bashrc` file (located at `~/.bashrc`) and remove the character sequence `@/h` from the `PS1` definitions.

4.5 LaTeX Compilation

In this section, I will describe how I set up streamlined compilation for editing LaTeX documents in Vim.

I began by creating a file as follows:

```
sudo touch /usr/local/bin/mupdf_autorefresh
```

To make the file executable, type the following into a terminal:

```
sudo chmod +x /usr/local/bin/mupdf_autorefresh
```

In that file, I wrote the following script:

```
#!/usr/bin/env bash

FILENAME=${1?Error: No pdf filename given. In vim, pass %:t:r} # name of
    pdf file
FILEPATH=${2?Error: No pdf file path given. In vim, pass %:p:r} # path of
    pdf file
NUMOPEN=$(pgrep -a mupdf | grep $FILENAME.pdf | awk {'print $1'} | wc -l) #
     number of open mupdf processes corresponding to $FILENAME

if [ "$NUMOPEN" -eq 0 ]; then # no windows open
        mupdf -r160 $FILEPATH.pdf # open pdf
else # multiple windows open
        pgrep -a mupdf | grep $FILENAME.pdf | awk {'print $1'} | xargs kill
     -HUP # update all open windows
fi
```

When called on a PDF file, this script will either open the PDF using MuPDF in the case that the PDF file is not already open in MuPDF, or if it is already open then the script will force MuPDF to update and display any changes in the PDF.

Next, I started editing Vim's configuration file (located at `~/.vimrc`) and added the following line:

```
:map <F4> :w<CR> :!latexmk -pdf % && latexmk -c % <CR> :!bash
   mupdf_autorefresh %:t:r %:p:r & <CR><CR>
```

This defines a keybinding such that pressing `<F4>` while editing a `.tex` file in Vim results in writing (saving) the `.tex` file, compiling the file using `latexmk`, clearing the temporary files from the project's directory, and calling the `mupd_fautorefresh` script described above (which updates the PDF preview).

Now, altough LaTeX will stop compilation in the case of a fatal error (and in turn, display the compilation output text), the above setup does not yield any output indicating non-fatal warnings (such as `overfull hbox` for instance). To enable notifications of non-fatal errors and warnings, I created a file as follows:

```
sudo touch /usr/local/bin/latexmk_warning_popup
```

To make that file executable, type the following into a terminal:

```
sudo chmod +x /usr/local/bin/latexmk_warning_popup
```

In that file, I wrote the following script:

```
#!/usr/bin/env bash

FILEPATH=${1?Error: No pdf file path given. In vim, pass %:p:r} # path of
    tex file

latexmk -pdf $FILEPATH.tex # compile latex document
NUMWARNINGS=$(cat $FILEPATH.log | grep -i "warning\|missing\|overfull\|
    underfull\|error\|!" | wc -w) # count number of warnings

if [ "$NUMWARNINGS" -ne 0 ]; then
        i3-sensible-terminal -e bash -c "cat $FILEPATH.log | grep --color -
    i 'warning\|missing\|overfull\|underfull\|error\|!'; bash" & # list
    warnings
fi

latexmk -c $FILEPATH.tex # clear temp files
```

When called on a `.tex` file, `latexmk` is used to compile the document. In the case that the compilation output text contains errors or warnings, a new terminal opens with the corresponding errors and warnings displayed and highlighted.

Next, I started editing Vim's configuration file (located at `~/.vimrc`) and added the following line:

```
:map <F5> :w<CR> :!latexmk_warning_popup %:p:r <CR> :!bash
    mupdf_autorefresh %:t:r %:p:r & <CR><CR>
```

This defines a keybinding such that pressing `<F5>` while editing a `.tex` file in Vim results in compiling the document and updating the PDF in the same way as described above for the `<F4>` keybinding, except that errors and warnings will be displayed in a popup terminal window.

4.6 Music Player Configuration

To configure cmus, press `7`. In the keybindings section, I remapped `p` to `player-pause`, `P` to `toggle continue`, `right` to `player-next`, `left` to `player-prev`, and then swapped the corresponding default keybindings.

I also changed some of the layout settings, as follows:

```
format_current          %a - %l (%y) -%3n. %t
format_playlist         %-15%a %l %= %3n. %-40%t %y %d %{?X!=0?%3X ? }
format_title            %a - %l (%y) - %t
format_trackwin         %3n. %t%= %y %d
```

5 Unfinished Business

In this section, I'll enumerate some of the aspects of installation and customization that I didn't finish, or that I couldn't get to work.

5.1 Customizing the Desktop

1. Get MuPDF to automatically open PDF documents full size (the same percentage as the LaTeX compiler) when opening from ranger.
2. Get Chromium "show in folder" button to use ranger when downloading files from online.
3. Get cmus controls to work in Polybar.
4. Customize the `.vimrc` file for LaTeX shortcuts.
   (a) Get clicked-based jump-to-line feature working.
   (b) Get `\ref` to auto-complete.
5. Get Print Screen button to work.
6. Customize colors, styles, and shortcuts.
   (a) Mutt.
   (b) Rofi.
   (c) cmus.
   (d) LightDM.
   (e) URxvt.
   (f) ranger.
7. Customize Matlab.
   (a) Customize file associations (ranger should open Matlab if not editing from Vim).
   (b) Make i3 shortcut to open Matlab.
   (c) Get Matlab to run scripts through terminal (and generate plots).

5.2 Surface Pro Functionality

1. Get cameras working, then get Ubuntu Howdy working.
2. Get pen button to open Xournal (see `freebib`'s script).
   (a) Sometimes Xournal must be opened twice before it recognizes the pen. Fix this.
3. Get auto-sleep with keyboard working.
4. Sometimes there is still audio crackling in i3. Fix this.
5. Fix trackpad issues.

(a) Natural scrolling and tap-to-click are turning off automatically (especially after hibernating). It seems like this is due to the keyboard disconnecting, causing `xinput` settings to be reset.

6. Fix auto-load issues.

   (a) Loading of programs in i3 at startup seems inconsistent. Possible fix: put flags on each program in the configuration file when initially calling them to open, and use those flags as identifiers in the `json` files. See Section 3 of the official i3 layout guide where they show instance matching for Emacs.