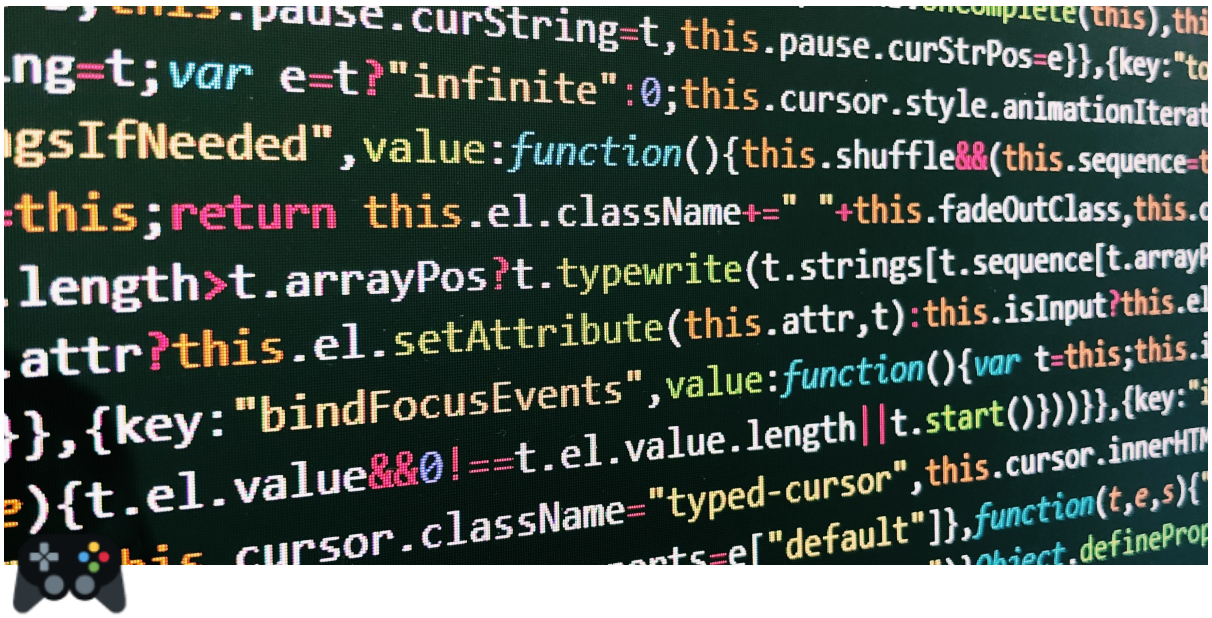

Lecture 10: Potpourri

Keyboard remapping

The most basic change is to remap keys. This usually involves some software that is listening and, whenever a certain key is pressed, it intercepts that event and replaces it with another event corresponding to a different key.

- Remap Caps Lock to Ctrl or Escape. We (the instructors) highly encourage this setting since Caps Lock has a very convenient location but is rarely used.
- Remapping PrtSc to Play/Pause music. Most OSes have a play/pause key.
- Swapping Ctrl and the Meta (Windows or Command) key.

Some software resources to get started on the topic:

- macOS - karabiner-elements, skhd or BetterTouchTool
- Linux - xmodmap or Autokey
- Windows - Builtin in Control Panel, AutoHotkey or SharpKeys
- QMK - If your keyboard supports custom firmware you can use QMK to configure the hardware device itself so the remaps works for any machine you use the keyboard with.

Daemons

Most computers have a series of processes that are always running in the background rather than waiting for a user to launch them and interact with them. These processes are called **daemons** and the programs that run as daemons often end with a `d` to indicate so. For example `sshd`, the SSH daemon.

In Linux, `systemd` (the system daemon) is the most common solution for running and setting up daemon processes. You can run `systemctl status` to list the current running daemons. Systemd can be interacted with the `systemctl` command in order to `enable`, `disable`, `start`, `stop`, `restart` or check the `status` of services (those are the `systemctl` commands).

`systemd` has a fairly accessible interface for configuring and enabling new daemons (or services). Below is an example of a daemon for running a simple Python app.

```
# /etc/systemd/system/myapp.service
[Unit]
Description=My Custom App
After=network.target

[Service]
User=foo
Group=foo
WorkingDirectory=/home/foo/projects/mydaemon
ExecStart=/usr/bin/local/python3.7 app.py
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Also, if you just want to run some program with a given frequency there is no need to build a custom daemon, you can use `cron`, a daemon your system already runs to perform scheduled tasks.

FUSE

FUSE (Filesystem in User Space) allows filesystems to be implemented by a user program. FUSE lets users run user space code for filesystem calls and then bridges the necessary calls to the kernel interfaces.

Some interesting examples of FUSE filesystems are:

- sshfs - Open locally remote files/folder through an SSH connection.
- rclone - Mount cloud storage services like Dropbox, GDrive, Amazon S3 or Google Cloud Storage and open data locally.
- gocryptfs - Encrypted overlay system. Files are stored encrypted but once the FS is mounted they appear as plaintext in the mountpoint.
- kbfs - Distributed filesystem with end-to-end encryption. You can have private, shared and public folders.
- borgbackup - Mount your deduplicated, compressed and encrypted backups for ease of browsing.

Backups

First, a copy of the data in the same disk is not a backup, because the disk is the single point of failure for all the data. Similarly, an external drive in your home is also a weak backup solution.

Synchronization solutions are not backups. For instance, Dropbox/GDrive are convenient solutions, but when data is erased or corrupted they propagate the change.

Some core features of good backups solutions are versioning, deduplication and security:

- Versioning backups ensure that you can access your history of changes and recover files.
- Efficient backup solutions use data deduplication to only store incremental changes and reduce the storage overhead.

- Regarding security, you should ask yourself what someone would need to know/have in order to read your data and, more importantly, to delete all your data and associated backups.

For a more detailed explanation, see 2019's lecture notes on Backups.

APIs

You will find that many of these lessons also extend to the wider internet. Most services online will have "APIs" that let you programmatically access their data.

Most of these APIs have a similar format. They are structured URLs, often rooted at `api.service.com`, where the path and query parameters indicate what data you want to read or what action you want to perform.

The response itself contains a bunch of other URLs that let you get specific forecasts for that region. Usually, the responses are formatted as JSON, which you can then pipe through a tool like `jq` to massage into what you care about.

Common command-line flags/patterns

Command-line tools often share some common features though that can be good to be aware of:

- Most tools support some kind of `--help` flag to display brief usage instructions for the tool.
- Many tools that can cause irrevocable change support the notion of a "dry run" in which they only print what they would have done, but do not actually perform the change. Similarly, they often have an "interactive" flag that will prompt you for each destructive action.
- You can usually use `--version` or `-V` to have the program print its own version (handy for reporting bugs!).
- Almost all tools have a `--verbose` or `-v` flag to produce more verbose output. You can usually include the flag multiple times (`-vvv`) to get more verbose output. Similarly, many tools have a `--quiet` flag for making it only print something on error.
- In many tools, `-` in place of a file name means "standard input" or "standard output", depending on the argument.

- Possibly destructive tools are generally not recursive by default, but support a "recursive" flag (often `-r`) to make them recurse.
- Sometimes, you want to pass something that looks like a flag as a normal argument. For example, imagine you wanted to remove a file called `-r`. Or you want to run one program "through" another, like ssh machine foo, and you want to pass a flag to the "inner" program (foo). The special argument `--` makes a program stop processing flags and options (things starting with `-`) in what follows, letting you pass things that look like flags without them being interpreted as such: `rm -- -r` or `ssh machine --for-ssh -- foo --for-foo`.

VPNs

A VPN, in the best case, is **really** just a way for you to change your internet service provider as far as the internet is concerned.

While that may seem attractive, keep in mind that when you use a VPN, all you are really doing is shifting your trust from you current ISP to the VPN hosting company. Whatever your ISP could see, the VPN provider now sees instead. Some VPN providers are malicious (or at the very least opportunist), and will log all your traffic, and possibly sell information about it to third parties.

Booting + Live USBs

Live USBs are USB flash drives containing an operating system. You can create one of these by downloading an operating system (e.g. a Linux distribution) and burning it to the flash drive. This process is a little bit more complicated than simply copying a `.iso` file to the disk. There are tools like UNetbootin to help you create live USBs.

Live USBs are useful for all sorts of purposes. Among other things, if you break your existing operating system installation so that it no longer boots, you can use a live USB to recover data or fix the operating system.

Docker, Vagrant, VMs, Cloud, OpenStack

Virtual machines and similar tools like containers let you emulate a whole computer system, including the operating system.

Vagrant is a tool that lets you describe machine configurations (operating system, services, packages, etc.) in code, and then instantiate VMs with a simple `vagrant up`. Docker is conceptually similar but it uses containers instead.

Github

There are two primary ways in which people contribute to projects on GitHub:

- Creating an issue. This can be used to report bugs or request a new feature. Neither of these involves reading or writing code, so it can be pretty lightweight to do.
- Contribute code through a pull request. This is generally more involved than creating an issue. You can fork a repository on GitHub, clone your fork, create a new branch, make some changes (e.g. fix a bug or implement a feature), push the branch, and then create a pull request. After this, there will generally be some back-and-forth with the project maintainers, who will give you feedback on your patch. Finally, if all goes well, your patch will be merged into the upstream repository.