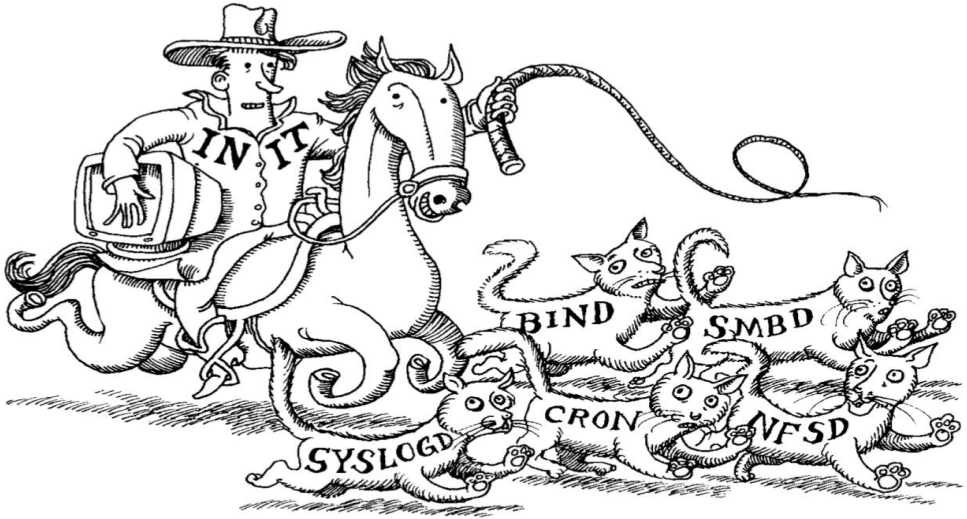
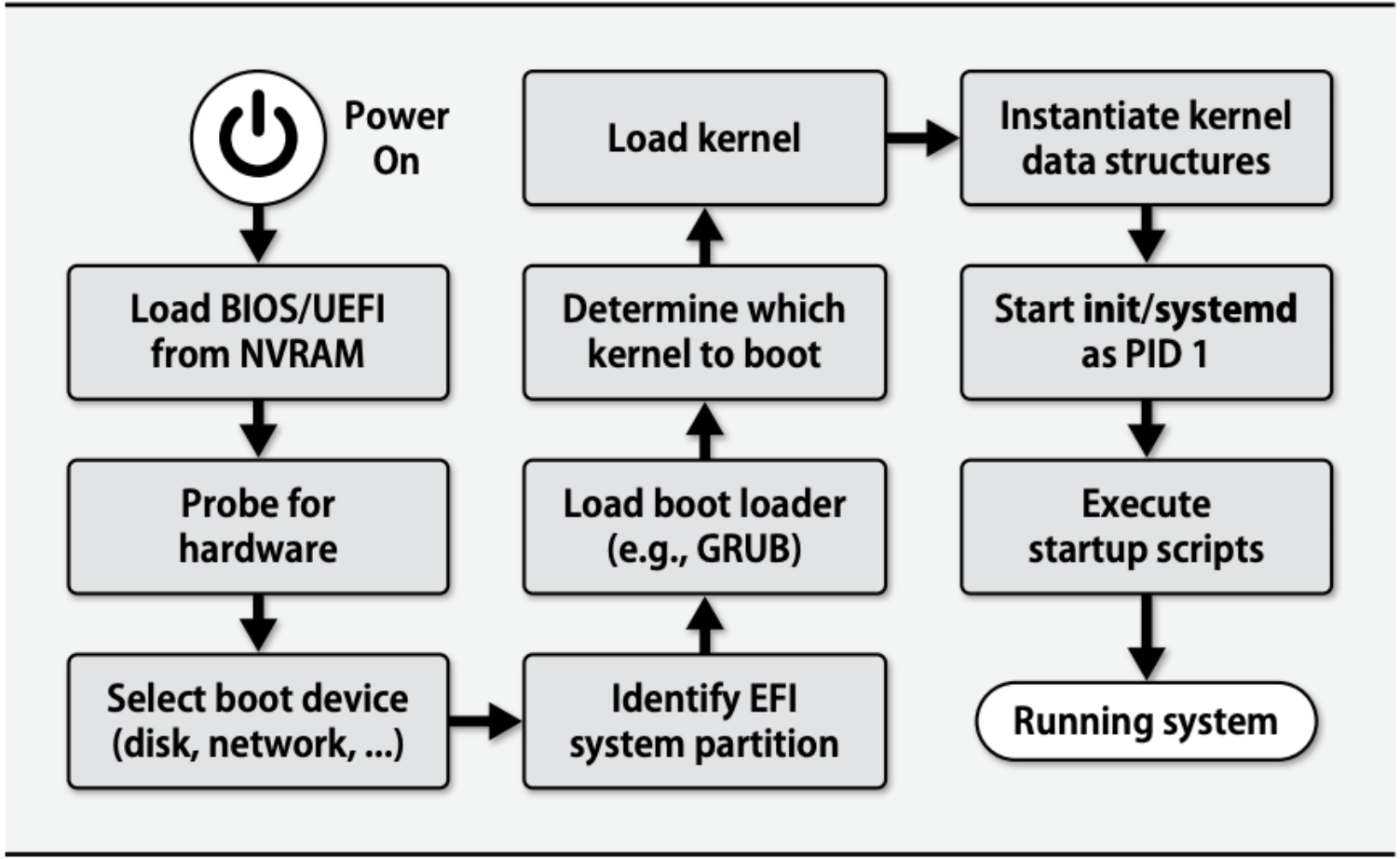
CS183

Instructor: Ali Davanian
(Slides were adopted from Brian Crites and Alireza Abdoli)

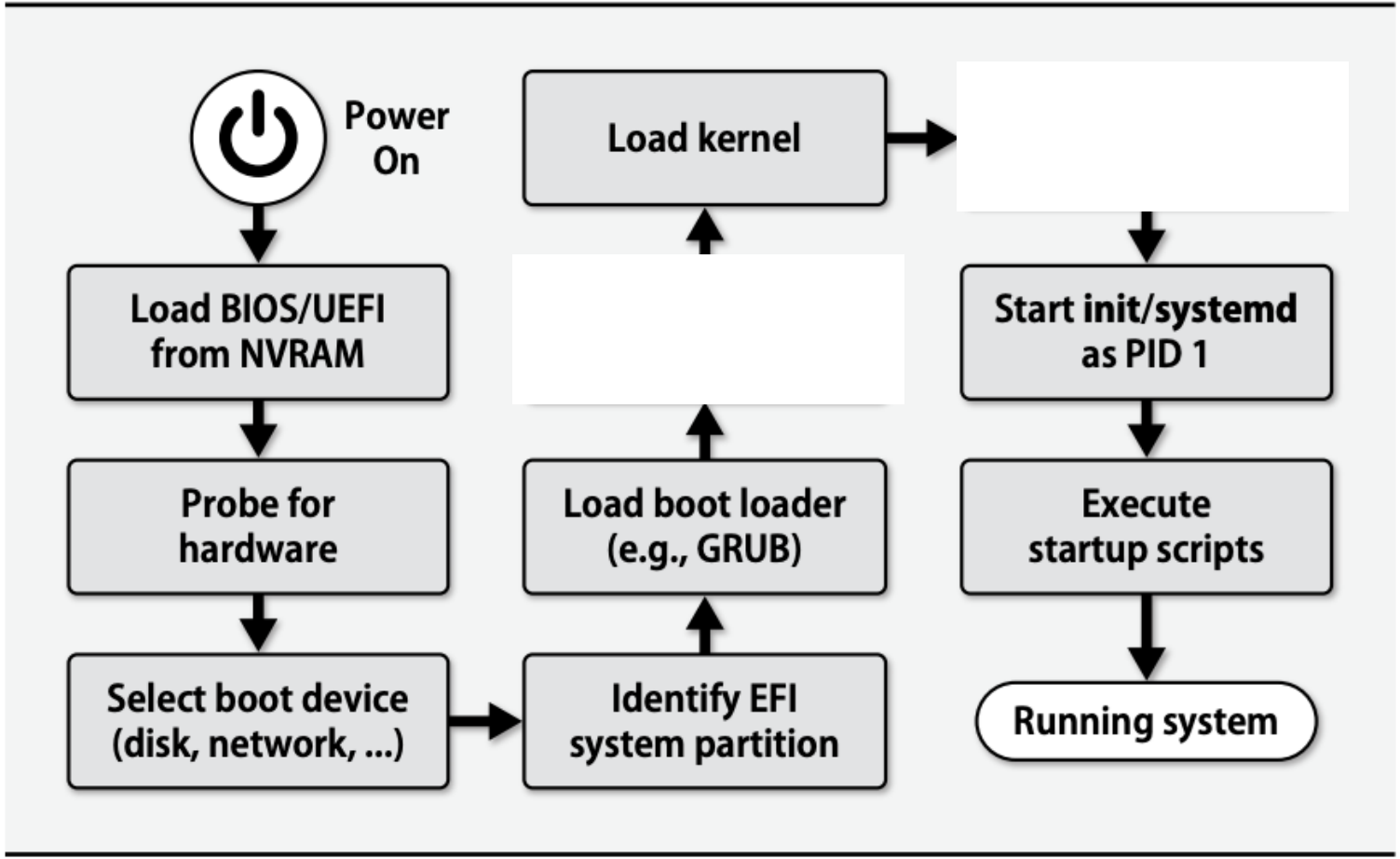


Linux Boot Systems

Linux & UNIX boot process

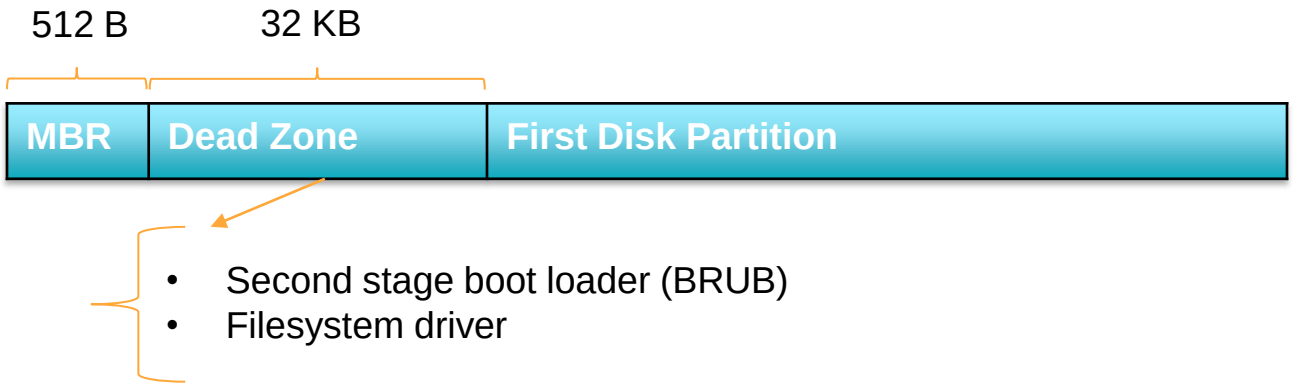


Linux & UNIX boot process



Three Booting Stages (Basic I/O System - BIOS)

(1) Master Boot Record (MBR): Points to the secondary boot loader
(2) Secondary Boot Loader
    (a) Usually GRand Unified Bootloader (GRUB) which lets you choose which partition to load as the OS
    (b) GRUB loads the secondary bootloader for the specific OS kernel
(3) OS Kernel: The core functionality for your operating system



Unified Extensible Firmware Interface (UEFI)

- BIOS is being replaced by Unified Extensible Firmware Interface (**UEFI**) which covers both MBR and secondary boot loader with some extra features:
  - It understands FAT file system
  - Less bootstrapping
  - UEFI has a formal API that allows accessing the system’s hardware and custom configurations

GRUB

- Understands **file systems** giving it access to enough memory to load the OS
- Reads a configuration file usually at /boot/grub(2)/grub.cfg
- Can be generated with the utility grub(2)-mkconfig
- Config is often re-created during OS updates, so you may need to take action to stop custom grub.cfg files from being overwritten

Common GRUB configuration options from /etc/default/grub

Shell variable name	Contents or function
GRUB_BACKGROUND	Background image[a]
GRUB_CMDLINE_LINUX	Kernel parameters to add to menu entries for Linux[b]
GRUB_DEFAULT	Number or title of the default menu entry
GRUB_DISABLE_RECOVERY	Prevents the generation of recovery mode entries
GRUB_PRELOAD_MODULES	List of GRUB modules to be loaded as early as possible
GRUB_TIMEOUT	Seconds to display the boot menu before autoboot

a. The background image must be a **.png**, **.tga**, **.jpg**, or **.jpeg** file.
b. Table 2.3 on page 38 lists some of the available options.

Kernel Booting (Init/Systemd)

- Kernel starts a single init or systemd process with a PID of 1
- This process spawns all other processes in the system (using fork)
- Options for booting specified in /etc/grub.d/40_custom

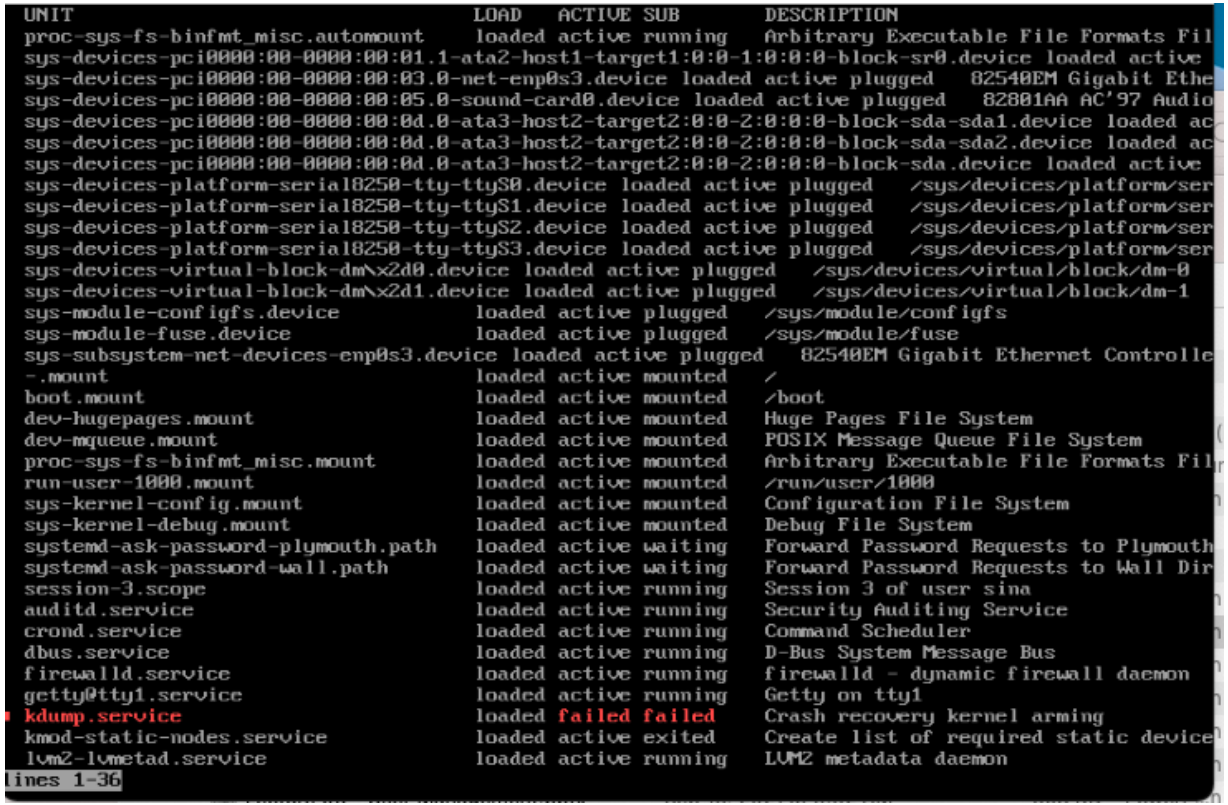
Examples of kernel boot time options

Option	Meaning
`debug`	Turns on kernel debugging
`init=/bin/bash`	Starts only the **bash** shell; useful for emergency recovery
`root=/dev/foo`	Tells the kernel to use **/dev/foo** as the root device
`single`	Boots to single-user mode

$ ps -aux

```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.3 125488  3876 ?        Ss   10:37   0:01 /usr/lib/systemd/systemd --switched
-root --system --deserialize 22
root           2  0.0  0.0      0     0 ?        S    10:37   0:00 [kthreadd]
```

Kernel has many components known as “UNIT”s



- Services
- Sockets
- Devices
- Mount points
- Partitions
- Timers
- etc.

Systemd

- Anything managed by systemd is known as a “unit”
- Each unit describes some boot item or a boot requirement
  - services, sockets, devices, mount points, partitions, timers, etc.
- systemd units are located at:
  - /lib/systemd/system
  - /usr/lib/systemd/system
  - /etc/systemd/system

```
[Unit]
Description=fast remote file copy program daemon
ConditionPathExists=/etc/rsyncd.conf

[Service]
ExecStart=/usr/bin/rsync --daemon --no-detach

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

Unit Files

- Describes dependencies between other units and targets
- Describes relative ordering between dependent units, allowing for parallelism
- Describes various modes of operation
- Can handle preparation and post-death cleanup
- Lots more...

```
[Unit]
Description=The nginx HTTP and reverse proxy server
After=network.target remote-fs.target nss-lookup.target

[Service]
Type=forking
PIDFile=/run/nginx.pid
ExecStartPre=/usr/bin/rm -f /run/nginx.pid
ExecStartPre=/usr/sbin/nginx -t
ExecStart=/usr/sbin/nginx
ExecReload=/bin/kill -s HUP $MAINPID
KillMode=process
KillSignal=SIGQUIT
TimeoutStopSec=5
PrivateTmp=true

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

Unit Files

```
[Unit]
Description=The nginx HTTP and reverse proxy server
After=network.target remote-fs.target nss-lookup.target

[Service]
Type=forking
PIDFile=/run/nginx.pid
ExecStartPre=/usr/bin/rm -f /run/nginx.pid
ExecStartPre=/usr/sbin/nginx -t
ExecStart=/usr/sbin/nginx
ExecReload=/bin/kill -s HUP $MAINPID
KillMode=process
KillSignal=SIGQUIT
TimeoutStopSec=5
PrivateTmp=true

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

- [Unit] Section
  - Starts with a description of the process
  - After lists other targets which this process requires to be able to run

Unit Files

```
[Unit]
Description=The nginx HTTP and reverse proxy server
After=network.target remote-fs.target nss-lookup.target

[Service]
Type=forking
PIDFile=/run/nginx.pid
ExecStartPre=/usr/bin/rm -f /run/nginx.pid
ExecStartPre=/usr/sbin/nginx -t
ExecStart=/usr/sbin/nginx
ExecReload=/bin/kill -s HUP $MAINPID
KillMode=process
KillSignal=SIGQUIT
TimeoutStopSec=5
PrivateTmp=true

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

- [Service] Section
  - Type defines its running condition, here forking means the daemon should keep running even though the process finishes
  - Daemon PID is recorded in PIDFile location, since forking means systemd won't have created the daemon directly
  - Exec lines show commands to run during various forms of execution (pre, start, reload, etc.)
  - KillMode and KillSignal state that a QUIT signal should terminate and clean up
  - PrivateTmp is for security reasons and puts the /tmp directory someplace users can’t freely access it

Unit Files

```
[Unit]
Description=The nginx HTTP and reverse proxy server
After=network.target remote-fs.target nss-lookup.target

[Service]
Type=forking
PIDFile=/run/nginx.pid
ExecStartPre=/usr/bin/rm -f /run/nginx.pid
ExecStartPre=/usr/sbin/nginx -t
ExecStart=/usr/sbin/nginx
ExecReload=/bin/kill -s HUP $MAINPID
KillMode=process
KillSignal=SIGQUIT
TimeoutStopSec=5
PrivateTmp=true

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

- [Install] Section
  - WantedBy links this unit with a target, so when the target is enabled this unit will be linked
  - This allows the target to stay relatively clean but to be extended from the outside

systemctl

- Allows you to investigate and modify systemd
- Systemd unit file describes what it does, systemctl enables/disable it to run at boot

Commonly used systemctl subcommands

Subcommand	Function
list-unit-files [*pattern*]	Shows installed units; optionally matching *pattern*
enable *unit*	Enables *unit* to activate at boot
disable *unit*	Prevents *unit* from activating at boot
isolate *target*	Changes operating mode to *target*
start *unit*	Activates *unit* immediately
stop *unit*	Deactivates *unit* immediately
restart *unit*	Restarts (or starts, if not running) *unit* immediately
status *unit*	Shows *unit*'s status and recent log entries
kill *pattern*	Sends a signal to units matching *pattern*
reboot	Reboots the computer
daemon-reload	Reloads unit files and **systemd** configuration

Unit file statuses

State	Meaning
`bad`	Some kind of problem within **systemd**; usually a bad unit file
`disabled`	Present, but not configured to start autonomously
`enabled`	Installed and runnable; will start autonomously
`indirect`	Disabled, but has peers in `Also` clauses that may be enabled
`linked`	Unit file available through a symlink
`masked`	Banished from the **systemd** world from a logical perspective
`static`	Depended upon by another unit; has no install requirements

Journald and journalctl

- Journald is a system paired with systemd that logs for all the units under systemd control
- Use journalctl to read these logs (located at run/)
- Notable flags: -b X allows you to go back to a previous boot (negative number) or a specific boot (boots hash) -u X allows you to look for a specific unit (service name)

```
$ journalctl
-- Logs begin at Fri 2016-02-26 15:01:25 UTC, end at Fri 2016-02-26
    15:05:16 UTC. --
Feb 26 15:01:25 ubuntu systemd-journal[285]: Runtime journal is using
    4.6M (max allowed 37.0M, t
Feb 26 15:01:25 ubuntu systemd-journal[285]: Runtime journal is using
    4.6M (max allowed 37.0M, t
Feb 26 15:01:25 ubuntu kernel: Initializing cgroup subsys cpuset
Feb 26 15:01:25 ubuntu kernel: Initializing cgroup subsys cpu
Feb 26 15:01:25 ubuntu kernel: Linux version 3.19.0-43-generic (buildd@
    lcy01-02) (gcc version 4.
Feb 26 15:01:25 ubuntu kernel: Command line: BOOT_IMAGE=/boot/vmlinuz-
    3.19.0-43-generic root=UUI
Feb 26 15:01:25 ubuntu kernel: KERNEL supported cpus:
Feb 26 15:01:25 ubuntu kernel:   Intel GenuineIntel
...
```

What to do if a system fails to boot?

- Don’t debug; just restore the system to a known-good state
  - Always possible assuming you have backups of your system
  - Depending on what you backup and what you need to restore, may be costly
- Bring the system up just enough to run a shell, and debug interactively
  - Use single-user, rescue, or emergency mode to bring up all little as possible before a shell
  - May not work in the case that the bug is very early in the boot
  - Doesn’t boot network, so requires physical access to the system
- Boot a separate system image, mount the sick system’s filesystem, and investigate
  - Requires the installation of a new system image in addition to the sick one
  - Gives you an external view of the system, which may make some issues more difficult to trace

Shutdown Procedures

- Shutting down a system is governed by three primary commands:
  - `halt`: performs necessary shutdown procedures, logs the shutdown, kills non essential processes, flushes cached filesystem blocks to disk (add `-p` flag to power off the system as well)
  - `reboot`: this performs a halt and then causes the system to boot again
  - `shutdown`: allows for scheduled halts and reboots and prints warnings to users that the system will soon become unavailable

Questions?

Additional Resources

Digital Ocean: Understanding Systemd Units and Unit Files