Linux System Administration and FSL10

David Horsley (NVI/NASA GSFC) Jonathan Quick (HartRAO)

TOW 2019

Goals

These notes are aimed at giving you

- an overview of Linux/Unix systems and their operations
- skills in configuring a system
- tools to diagnose hardware and software failures
- some specific info and debugging tips for Field System machines

This will be somewhat basic but is not a first introduction to Linux. You are expected to be familiar with a shell and a text editor (eg, **nano**).

At the end, we will discuss changes in the upcoming FSL10

Notation

Program and file names will be printed like this: `ls`, `/etc/fstab`, ...

To distinguish user input from output, it will start with a “`>`”, eg:

```
> ls /
bin
dev
...
```

(Don’t enter the “`>`” character)

Privileged user input (eg. as `root`) will start with `#`, eg:

```
# reboot
shutting down for system reboot
```

Background

Linux kernel

Linux is an operating system kernel [1]

At its most basic level, a kernel is software that manages the interaction between a computer's hardware and the running programs.

It allows multiple programs to share the computer's hardware and provide a secure separation between them.

It also provides programs with a (mostly) hardware independent interface to resources.

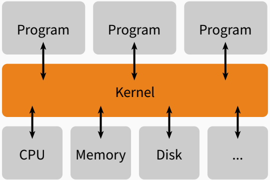
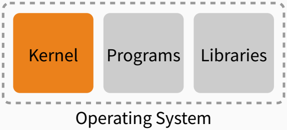


Figure 1: Kernel's Role

[1] *ker-nel*: the central or most important part of something. syn: essence, core, heart.

Operating System



To make usable *operating system*, you must combine the kernel with system and application software. Things like **`ls`**, **`mv`**, **`mount`**, as well as high level things like desktops.

Typically the core set of tools are the GNU, which make the *GNU/Linux Operating System*.

Often this is just called “Linux”.

The Linux kernel is also used by some non-GNU or even in non UNIX-like OSs (Android, TVs, routers, cars, ...)

You’re probably more than a few metres from a Linux kernel!

Linux Distributions

A Linux *distribution* is a particular packaging of the Linux OS. They provide the kernel, a suite of system and application software, and tools to manage it all.

Different distributions ("distros") can have very different system tools and philosophy, but all have a core set of UNIX programs.

Some Linux Distributions

- **Debian**: One of the oldest. Large repository of packages. Focus on free (libre) software and stability. **Supported distribution for Field System computers**.
- **Ubuntu**: Fork of Debian, focus on ease of use and commercial support. Provides non-free (propriety) drivers by default.
- **Red Hat Enterprise Linux**: focus on enterprise and server use.
- **Arch Linux**: rolling release, always latest stable versions of software. Designed to be minimal and hands-on.
- **GParted Live**: boots off a CD or USB drive and allows you to partition the system drives. Handy if you've broken your system!
- ... and hundred more!

Unix

GNU/Linux is just one part of a big family of UNIX-like OSs.

What we cover here will also apply to varying extent to other Unixes.

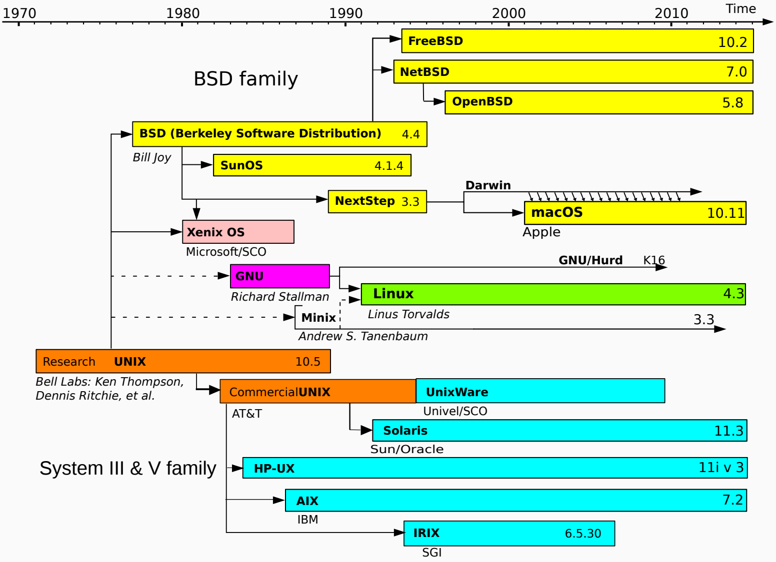


Figure 2: Timeline of Unix and Unix-like OSs. Linux and macOS (formally OS X) are the most common.

The State of Linux OSs

Most of the big GNU/Linux operating systems using a software suite called **systemd**.

- systemd is more than just an init system.
- it is intended to be a whole unifying **system layer** of the OS
- More components of the OS are being shifted into the systemd universe, but for now old UNIX ways still exist in parallel or at least are emulated by systemd

For a lot of what we will discuss, there is a more pure systemd way

- or if there isn't, it's probably planned
- today we will just cover the init system and service management.

The next version of this talk may be different.

Getting Help

RTFM: read the “friendly” manual

`man` (short for “manual”) provides extended documentation on tools and libraries on your system. Use it!

Try:

```
> man man
```

`man` pages are split into different sections. Sometimes you will need to specify the section when the page name alone is ambiguous, e.g.:

```
> man crontab
> man 5 crontab
```

(Section 5 is for file types)

Often you will see the section in parentheses after the page. E.g. `crontab(5)`

Other On-System Resources:

`apropos` [2] searches the **`man`** pages — forgot a command, use this.

Often the **`-h`** flag will provide short help.

[2] *ap-ruh-poh*: regarding/concerning

Other Resources

Field System Specific (found in `fs/misc`):

- FSL9/10 Installation Guide
- FSL9/10 RAID Guide

Debian Specific:

- Administrator's Handbook — Also see the older Wheezy version
- Reference Card — One page you can print and put beside your PC
- Bug Tracking System

Arch Linux Wiki — even if you don't use Arch Linux!

Google — Try writing in the program and the error you see on screen.

Basic Sysadmining

The Filesystem

Linux, as a UNIX-like OS, has a single *virtual* file system. All accessible files are somewhere in this tree, even if the they are on a different physical device.

The “root” of the tree is denoted by “**/**”

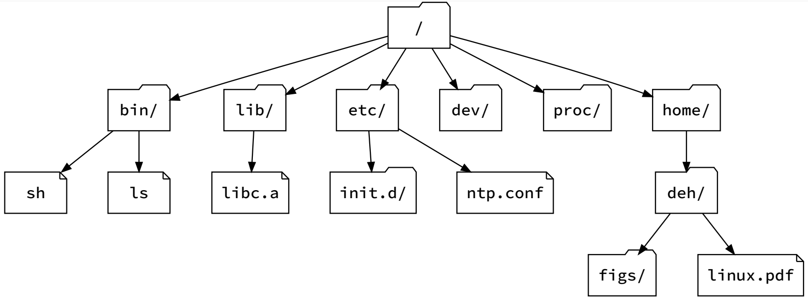


Figure 3: Unix virtual file system

Filesystem Info

- **`ls`** — list contents of a directory

  Extra info with **`-l`** flag.

- **`df`** — report file system disk space usage
- **`du`** — estimate file space usage

  Without arguments list all files and subdirectories. More useful is with flags:

  ```
  > du -hs
  ```

  List the usage of the current pathin human readable format (**`-h`**)

- **`lsblk`** — list block devices (physical and virtual disks)
- **`mount`** — List mounted file systems and mount flags. Also used for mounting other file systems — more on this later.

Users and Permissions

Linux as inherits Unix's concept of *users* and *groups*.

All files and directories ("folders") have an owner user and an owner group, as well as permissions associated with these:

- the owner user (**u**),
- the owner group (**g**),
- and other (**o**).

For each category of user, there are three fields or "bits" which control control if that type of user can:

- read (**r**),
- edit (**w**),
- or execute (**x**) the file. (**x** on a directory allows a user to access its contents)

```
> ls -l /home/deh/tow2019/
-rw----r-- 1 deh tow      3422 Apr 25  2019 linux.pdf
drwxr-xr-x 3 deh deh      4096 Apr 15  2019 figs
```

Changing Permissions

To change permissions use **chmod**

```
> cd /home/deh/tow2019/
> ls -l linux.pdf
-rw-r--r-- 1 deh tow      3422 Apr 25  2019 linux.pdf
> chmod u+x linux.pdf
> chmod g+w linux.pdf
> chmod o-r linux.pdf
-rwxrw---- 1 deh tow      3422 Apr 25  2019 linux.pdf
```

Special Permissions Bits

- **`u-s`** — setuid (set user identity).

  Makes an executable file run with the permissions of the user. (This is how **`passwd`** can change the usually inaccessible files to change your password)

  No effect on directories.

- **`g-s`** — setguid (group id)

  Similar to setuid.

  For directories, put files created in the directory into the same group as the directory, no matter what group the user who creates them is in

- **`-t`** – “save program text on swap device”

  For directories, prevent users from removing files that they do not own in the directory

Superuser

The “superuser” `root` can bypass all these permissions. (Apparently named because it’s the only user that can write to the root directory)

`root` can delete, start, and stop anything; erase hard drives, etc. Be careful!

Most system files can be read all users (like `oper`), but only written to by `root`

To login as `root` from a regular account, use:

`su -`

Which either stands for “set user” or “superuser.” The flag “`-`” gives a login shell. You must enter `root`’s password.

If you want an independent session as `root`, change to a virtual console (eg, by pressing Ctrl-Alt-F1) and login as root. **Do not** start an X Windows session as `root`.

Sudo

`sudo` (superuser do) allows a user to run one command with root privileges, eg:

```
> sudo less /var/log/kern.log
```

`sudo` can be configured to give subset of root privileges to users. Eg, only allow certain commands to be run as root. `sudo` is preferred in some environments for auditing purposes.

For some OSs (eg Ubuntu and macOS), this is the default way to get `root` access.

The `sudo` package may not be installed by default. To install

```
# apt-get install sudo
```

`/etc/sudoers` is the access list

The password requested is the *user's* password, not `root`'s. By default `sudo` keeps the current session authorized for 15min — handy!

Standard Linux Directories

Most Linux distros follow the Filesystem Hierarchy Standard. Other Unixes are similar but not identical.

Path	Contents
`/bin`	essential programs (binaries)
`/lib`	essential application libraries
`/etc`	configuration files
`/usr`	"UNIX system resources". Nonessential resources.
`/tmp`	temporary files, typically cleared at shutdown
`/home`	user home directories
`/root`	the `root` user's home directory
`/boot`	boot files (optional)
`/sbin`	system administration programs
`/mnt`	temporary mount points
`/var`	contains "variable" data

`/usr` Subdirectories

`/usr/` also contains **`bin`** and **`lib`** which store non critical programs and libraries. This is most of them.

There is also **`/usr/share/doc`** which stores documentation.

`/var` Subdirectories

`/var` contains variable data that changes while system is running:

- **`/var/log`** — run-time log files
- **`/var/spool`** — queued files (e.g. Printer)
- **`/var/mail`** — mailboxes
- **`/var/lock`** — “lock” files to ensure only one copy of a program is running
- **`/var/run`** — interface
- **`/var/tmp`** — similar to **`/tmp`** but typically with longer lifetime
- **`/var/cache`** — cache of downloads and computations. Can be cleared without loss.
- **`/var/lib/dpkg/info`** – status of installed software

Everything is a file

A big idea of Unix, is that “everything is a file”

Devices plugged into your computer, system processes, interfaces to running programs, everything![3] are all represented as files in the (virtual) filesystem

[3]well not quite everything; see “Plan 9 From Bell Labs”, an intended successor to Unix, for an OS that really, *really* made everything a file, including CPUs on on someone else’s computer!

Some Nonfile files

Some directories that are all purely virtual:

- `/proc` — process information
- `/dev` — device files which provide a file interface to physical devices.
- `/sys` — info about devices and high-level interface to some components.

`/proc` and processes

A **`process`** is a running program

`/proc` contains a directory for every process, which contains files that can be read to retrieve info the process.

You can, eg, list all the files open by a process **`ddout`**:

```
ls -l /proc/$(pgrep ddout)/fd
```

`/proc` also contains files used for querying the system information.

Eg.

- `/proc/cpuinfo` — info on the CPU(s) of the system
- `/proc/meminfo` — free and used memory in the system.

  Typically you will use `free` for this. Particularlly useful is the `-h` flag to `free` which display the output in human readable format (mega/giga bytes)

Inspecting processes

Usually you also don't inspect `/proc` manually to view process information, instead use a tool. For example, to view a dynamic display run

```
top
```

Press '`x`' to sort and '`<`' and '`>`' to select the column (eg. to `%CPU` or `%MEM`)

Print of current running processes to stdout

```
ps -ef
```

Print the process tree:

```
ps axjf
```

Note some information is only available to `root`.

Search through processes

```
pgrep -a <name>
ps -ef | grep <name>
```

Killing processes

To send a “terminate” signal to a process: (`<pid>` is the process identifier — found, for eg, with `ps` or `top`)

```
> kill <pid>
```

This will instruct the process to gracefully shutdown. Regular users can kill only their own processes, `root` can kill anything.

For more drastic situations, send a “kill” signal (see `man 7 signal`)

```
> kill -KILL <pid>
> kill -9 <pid>
```

This instructs the kernel to deal with the process itself, without a chance to cleanup.

To kill all programs matching `<name>`:

```
> pkill <name>
> killall <name>
```

Careful with these (eg “fs” may match a lot of processes you don’t want to kill!)

Some Devices in `/dev`

- **`/dev/sda`**, **`/dev/sdb`**, … — SATA disks
- **`/dev/hda`**, **`/dev/hdb`**, … — Older IDE disks
- **`/dev/ttyS01`**, … — Serial ports.
- **`/dev/audio`** — Audio port

There are also device files that are not physical ones, eg:

- **`/dev/zero`** — An infinite number of zeroes
- **`/dev/null`** — The “bit bucket”, send data here you want to disappear.
- **`/dev/random`**, **`/dev/urandom`** — a stream of random bits. (**`random`** can block, **`urandom`** will not)

Make white-noise:

```
# cat /dev/urandom > /dev/audio
```

“Securely” wipe a disk with noise:

```
# cat /dev/urandom > /dev/sda
```

Hard Disks

The whole disk

- **/dev/sda**
- **/dev/sdb**, **c**, **d** etc.

(**/dev/hda** etc for IDE)

Typically disks are these are *partition* into smaller segments. A partition is a contiguous part of the whole disk.

These show up in Linux as

/dev/sda1, **2**, ...

Partitions

These are use to:

- Separate user files and system files — as done in FS PCs. This allows the system partition to be wiped without affecting user data.
- Have different OSs (like Windows and Linux) on the same disk
- To have boot files accessible to older BIOS's by keeping them below the 1024 cylinders boundary

Partition tables are data at the start of the disk and describe the partition boundaries to the OS.

There are two common standards for this:

Master Boot Record (MBR)

Also sometimes called DOS partitions.

Old format

- Limited to 2 TB
- Limited to 4 partitions per disk.
    - This can be overcome by creating a special *extended partition* which holds up to 16 "logical drives". In Linux, these will show up as `/dev/sda5`, ...
- Can use `fdisk` manipulate partitions

GUID Partition Table (GPT)

New format

- Limited to 9 ZB
- Limited to 2^{64} partitions.
- Can use `gdisk` to manipulate

`parted` or its GUI counterpart `gparted` provide convenient interfaces for editing partitions. Changing partitions can **destroy** all the data on the disk!

MD — Linux Software RAID

RAID (Redundant Array of Independent/Inexpensive Disks) allows you to combine disks to help protect data from disk failures.

md (Multiple Device) is Linux's software RAID layer.

This allow you to combine block devices (disks, partitions, memory, ...) into one.

- **RAID0:** split ("stripe") data over multiple disks. **not really RAID** (no *R*). Use if you have lots of data if you want to access fast and you can afford to lose it.

  Data can survive after 0 failures. Lose 0 disks of space.

- **RAID1:** copy ("mirror") data on multiple disks.

  With n disks, data is safe after $n-1$ failures. Lose $n-1$ drives of space.

- **RAID5(6)**: uses a parity disk/partition to make a (fault tolerant) from many disks. Can survive 1 (2) disk failures.

RAIDs are presented as devices at `/dev/md*`

Also seen via `lsblk`

LVM

Logical Volume Management is a further layer in the Linux kernel.

Allows you to make “virtual” volumes on top of block devices, typically RAIDs.

Useful as you can’t partition a RAID device.

Can be use to make a **JBOD** array (just a bunch of disks).

Managing RAIDS

- **`md`** RAIDs are managed with **`mdadm`**
- **`md`** RAID state can be found in **`/proc/mdstat`**

File systems

To use a disk/partition/RAID to store files, you must create a *file system*

- This is book keeping data. It determines things like how files names, directories, modification times, permissions, as where the actual data are stored.

The current "native" format for Linux is the *fourth extended filesystem* **`ext4`**

To create a filesystem on a device:

- Hard disk partitions: **`mkfs.ext4 /dev/sda1`**
- Raid volumes: **`mkfs.ext4 /dev/md0`**

This is called *formatting* as it sets the format of the data. Note this effectively erases the device

Other File systems

Linux has extensive support for "foreign" file system types

Windows FAT partitions can be used as `vfat`

Network File System `nfs`, Windows `ntfs`, macOS `hfs+`, others...

Mounting

To access the files on the disk (partition/RAID/...), the filesystem must be *mounted*.

Mounting to a mount point (-directory)

```
# mount /dev/sda1 /home
```

(By default `mount` will try to detect the filesystem type, but you can explicitly set it with `-T`.)

Mount points are normal directories. Mounting hides the old directory contents.

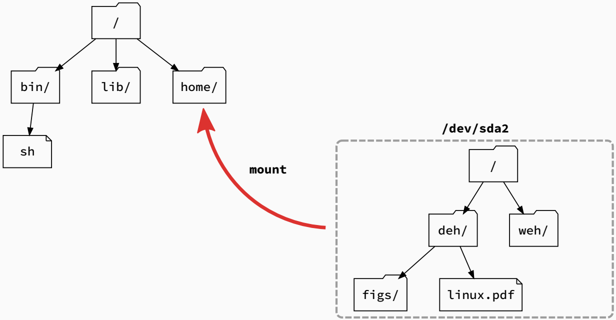


Figure 4: Before mounting

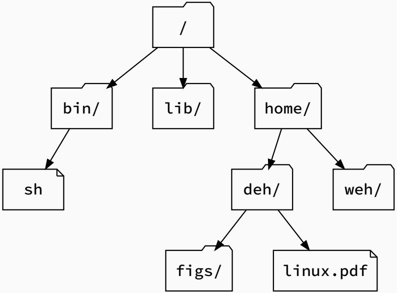


Figure 5: After mounting

Managing mounts

See what partitions are mounted (displays information from **`/proc/mounts`**) with

```
mount
```

Unmount with **`umount /mnt`**

Boot time mounts in **`/etc/fstab`**, mounted with **`mount -a`**. Eg:

```
/dev/sda3 /usr2 ext2 defaults 0 2
```

(see **`man fstab`**)

Unmounting is necessary before:

```
fsck, mkfs.ext3, fdisk, tune2fs
```

These directly alter file system / partition structures!

The Root Partition

The partition/device mounted as `/` is given to the kernel by the bootloader (GRUB)

Other partitions are mounted as listed in the `/etc/fstab` file (found on the `/` partition)

Network Info

`ip` — show / manipulate routing, devices, policy routing and tunnels (replaces **`ifconfig`**, **`route`**, and **`netstat`** for Linux)

Useful commands:

List IP address

```
> ip address  # ip a
```

List routes

```
> ip route    # ip r
```

`dig` — DNS lookup utility (alternative to **`nslookup`**)

User Info

w, who — Users logged in and what they are doing

Network Configuration

Filename	Description
/etc/hostname	Has name and IP address of this computer
/etc/hosts	
/etc/network/interfaces	The details of all available network interfaces
/etc/resolv.conf	Has the IP addresses of DNS name server(s)

Network Protection “tcpwrappers”

During boot, the “Internet super daemon” **`inetd`** is started

/etc/inetd.conf lists the services (TCP/UDP port numbers) **`inetd`** will listen to

When a connection from the outside is made, **`inetd`** runs the command listed in **`inetd.conf`**. This has been superseded by systemd sockets.

For almost all services, this is the **`tcpd`** wrapper which:

- First checks restrictions
- If allowed, starts the real service executable

tcpwrappers is not a firewall, it is an access list.

`/etc/hosts.allow` & `/etc/hosts.deny`

Have quite complex syntax (see `man 5 hosts_access` for details)

Effective only for entries with `tcpd` in `/etc/inetd.conf`

- Plus a couple of stand-alone server programs into which there is special support coded in
- For example the X server doesn't obey these!

`/etc/hosts.deny`:

```
ALL: ALL
```

`/etc/hosts.allow`:

```
ALL: .foobar.edu EXCEPT terminal.foobar.edu
```

- Executable names!

Further Editing in `/etc`

Disabling user accounts for logins

- Just replace the password in `/etc/passwd` with a `*`, eg:

  `user:*:500:500:...`

X configuration is now auto-generated. Use

`dpkg-reconfigure xserver-xorg`

etc.

Printers

CUPS printer daemon is configured in `/etc/cups/cupsd.conf`

Easiest configuration is using the CUPS web interface:

- Navigate to the URL http://localhost:631/

Background Process

Periodical Jobs with `cron`

The `cron` runs in the background with 1 min resolution, starting timed jobs

Debian's configuration files

`/etc/cron.d`

- Precisely timed jobs
- Special file format

  /etc/cron.daily, /etc/cron.weekly, /etc/cron.monthly

Plain shell scripts for periodical chores (like deleting old log files)

```
# Run queue every 5 minutes
*/5 * * * * /usr/sbin/exim -q >/dev/null 2>&1
```

`man 5 crontab`

Run once in the future

`at` is similar to `cron`, but is for one-off jobs, eg.

```
at 1pm
at today +2 hours
at 1135 jan
```

Daemons

Daemons[4] are background processes, not attached to a user's terminal.

Services on Linux/Unix are provided by daemons. These include things like

- Network time (**`ntpd`**)
- Network configuration (**`dhcpd`**)
- Secure (remote) Shell (**`sshd`**)
- Mail (**`exim4`**)
- ...

[4] From Maxwell's Demon. The Ancient Greek δαίμων, unlike the Christian "demon", is benevolent or benign being.

How to daemons come into being?

(We're not going to talk about The Silmarillion)

Daemons are generally started and managed by the *init* system.

Daemon may depend on services provided by other daemons or on machine state, so init can be tricky.

Eg, `ntpd` should be started once the network is ready, the graphical login manager should be started after the graphics system is ready

Linux PC-Level Startup

(On IBM Compatible PCs)

1. The motherboard performs a Power-On Self Test (POST)
   - Checks for required hardware: CPU, RAM, ...
   - Historically this was done by the BIOS (basic i/o system) now UEFI.
2. Motherboard then looks for bootable disks (Master Boot Record (MBR) or GPT)
   - For Linux systems, this means a partition contains GRUB, the GRand Unified Bootloader,
   - Order of disks can be set in BIOS

3. GRUB is loaded off the disk and starts by showing the boot menu. When you select your OS, the Linux kernel and other resources the early stage kernel will need into memory an jumps to the kernels entry point

4. Linux kernel starts, checks hardware, then attempts to locate the “root partition” This becomes the root (`/`) of the file system

5. Once `/` has been mounted (read-only), the kernel starts **`/sbin/init`**. As process #1 (PID 1), the grandparent of all processes

At this point, there a no disks mounted (except the read only initial partition). It’s **`init`**’s task to take this bare bones system to a usable state.

The state of `init`

What happens from here things vary between OSs/Distros.

- Until recently, (pre-2012) most Unixes used System V (SysV) style **`init`**.
- Most major Linux distributions have changed to **`systemd`**, which has a different model to support concurrent boot and more complex features.
- Others have also moved to OpenRC for similar reasons, which maintains a more SysV model.

We will cover systemd here, as this is the init system in use by Debian Stretch

If you have an older OS, you may need to lookup documentation of SysV init.

systemd targets

- systemd starts by loading the default *target* for the system
- In systemd, a *target* is a collection of services (daemons) and it can depend on lower level targets.
- For instance, a common target is the “graphical.target”. “multi-user.target” is another common target for headless systems.
- The default target is described in **`/etc/systemd/system/default.target`**, which is typically a symlink to a file in **`/usr/lib/systemd/system/`**

For example the “graphical.target” file looks like:

```
[Unit]
Description=Graphical Interface
Documentation=man:systemd.special(7)
Requires=multi-user.target
Wants=display-manager.service
Conflicts=rescue.service rescue.target
After=multi-user.target rescue.service rescue.target \
      display-manager.service
AllowIsolate=yes
```

systemd targets cont.

Targets are similar to SysV run levels, however they are much more versatile, as multiple targets can be active at once.

You can see the currently active targets with

```
systemctl list-units --type=target
```

Eg ...

```
UNIT                      LOAD   ACTIVE DESCRIPTION
basic.target              loaded active Basic System
bluetooth.target          loaded active Bluetooth
cryptsetup.target         loaded active Encrypted Volumes
getty.target              loaded active Login Prompts
graphical.target          loaded active Graphical Interface
local-fs-pre.target       loaded active Local File Systems (Pre)
local-fs.target           loaded active Local File Systems
multi-user.target         loaded active Multi-User System
network-online.target     loaded active Network is Online
network-pre.target        loaded active Network (Pre)
network.target            loaded active Network
nfs-client.target         loaded active NFS client services
nss-lookup.target         loaded active Host and Network
                                            Name Lookups
nss-user-lookup.target loaded active User and Group
                                            Name Lookups
paths.target              loaded active Paths
```

```
remote-fs-pre.target   loaded active Remote File Systems (Pre)
remote-fs.target       loaded active Remote File Systems
rpcbind.target         loaded active RPC Port Mapper
slices.target          loaded active Slices
smartcard.target       loaded active Smart Card
sockets.target         loaded active Sockets
sound.target           loaded active Sound Card
swap.target            loaded active Swap
sysinit.target         loaded active System Initialization
time-sync.target       loaded active System Time Synchronized
timers.target          loaded active Timers
```

Managing services with systemd

Services are the most common unit in systemd.

Typically they are provided by a daemon, which is managed by systemd

`systemctl` is the all purpose tool for managing systemd

Start services:

```
systemctl start sshd
```

Stop services:

```
systemctl stop sshd
```

Enable (on startup) :

```
systemctl enable sshd
```

systemd topics we didn't cover

As I mentioned, there's more to systemd we haven't covered here, including:

- sockets
- devices
- mounts and automounts
- paths
- timers (which can be used as a cron-like job scheduler)
- snapshots
- slices (used to group and manage processes and resources)
- journald

Maybe next time I give this talk we will replace all the Unix stuff with systemd.

Debian Sysadmin

APT — Advanced Packaging Tool

`apt` is the interface to Debian's package manager.

- Tracks package availability across multiple archives and releases
- Allows installation by package name directly

Replaced `dselect`

APT commands

Installation and removal:

```
apt install <name>
apt remove <name>
```

Update **apt**'s package list (sync with the servers):

```
apt update
```

Upgrade

```
apt upgrade          #all out-of-date packages
apt upgrade <name>
```

(Note: on old Debian based systems, need **apt-get**)

Search:

```
apt search
```

(Note: on old Debian based systems, need **apt-cache**)

Dpkg

`dpkg` is the lower-level system, accessed by **`apt`**

Debian's basic package tool

- Can install and remove **`.deb`** packages directly
- Knows about package dependencies but not about package archives and availability of updates

Keeps installed state in **`/var/lib/dpkg/info`**

`<name>.list, <name>.postinst`

All package installation, basic setup and removal is handled by **`dpkg`**

```
apt-cache --installed rdepends git
```

APT and Security Updates

`apt` also tracks security update availability at security.debian.org

Use `apt-get update` to reload package availability then `apt-get -u upgrade` to see what upgrades are currently available

`fsadapt` in FS Linux 9 installs automatic `cron` script based on this to warn about upgrades

Hardware Problems

Disks Failure

HDD and SSD failure is probably the most common problem nowadays.

HDDs failure mostly determined by age with ~2-4% Annualized Failure Rate (AFR) [5]

SSDs have a much lower AFR, but wear with number or writes. Wear is indicated by slowly increasing Uncorrectable Error Count.

SATA bus, cabling, connectors, terminators — Show up as nondeterministic disk failures

[5] Backblaze 2016 Stats https://www.backblaze.com/blog/hard-drive-benchmark-stats-2016/

Symptoms:

- Clicking or scratching sounds from the disk
- Unreadable blocks (see `/var/log/kern.log`)
- Increase rapidly over time → backup quickly

Disk Warning Signs

```
# smartctl -a /dev/sda
```

This gives you “S.M.A.R.T.” codes. Most important ones are: [6]

- SMART ID 187 (0xBB): Reported Uncorrectable Errors
  - HDD: 0: good; >0: replace
  - SSD: a few is ok; rapidly increasing is bad
- SMART ID 5 (0x05): Relocated Sectors Count
  - 0: good; 1-4: keep an eye on it; > 4: replace
  - SSD: a few is ok; rapidly increasing is bad
- SMART ID 188 (0xBC): Command Timeout
  - 1-13 keep an eye on it, more than 13 replace
- SMART ID 197 (0xC5): Current Pending Sector Count
  - 0: good; 1 or more: replace
- SMART ID 198 (0xC6): Uncorrectable Sector Count
  - 1 or more replace

[6] Backblaze Blog https://www.backblaze.com/blog/hard-drive-smart-stats/

Overheating

Fairly Common. Fans last 3–5 years. Dust can be a problem.

Symptoms:

- Modern motherboards will shutdown the computer if the CPU gets too hot (~100 C)
  - You may get an audible alarm
  - A good sign is if the system operates for a few minutes before shutting down.
  - You can check temperates and alarms in BIOS
- Older PCs may behave erratically

Causes:

- Clogged or broken CPU heat-sink fan
- Bad thermal connection between CPU and heat-sink
- Bad airflow inside case

Fix:

- Clean case and fans
- Replace fans

Motherboard Problems

Less likely. Motherboards can last 5-10 years if survive first year.

Symptoms:

- Hard to diagnose
- Does not POST
- Random Reboots
- Peripherals

Causes:

- Overheating
- Age

Fix:

- Replace

Memory

Less likely. Similar to motherboards, if not DOA, probably last 5-10 years.

Symptoms:

- Random program crashes
- Random reboots

Checks:

Newer Field System machines come with ECC RAM. To check status use EDAC (Error Detection And Correction) utils (install `edac-utils`):

```
# edac-util
```

Add a `memtest86+` to your GRUB menu

```
# apt-get install memtest86+
```

Reboot to it and let run for several hours.

Field System Linux 10

Field System Linux 10

Field System Linux is the supported OS for FS operations.

- Mostly this is a standard Debian install with some predefined packages and checkout procedures.
- We also define supported hardware and a backup schedule

FSL10 will be the next supported OS, based on Debian 9 (“stretch”)

- Currently testing, Expected release summer 2019
- Hardware support is getting a lot easier, but we will need some brave stations to test.

Changes:

- Better packages installed by default!
- RAID changes:
  - Single RAID block device rather than 3, with LVM partitions on top.
  - Fix some issues with swap partition.
  - Also allow resizing of root if needed
- Dual 32/64bit (x86) architectures
- Support for compiling with `gfortran`
- SysV init scripts converted to systemd units
  - Recommend doing the same for any station init scripts.
- Additional security required at NASA stations provided as option for everyone else.

Converting station code to 64bit

The biggest obstacle for compiling FS/station code on `x86_64` is the size of `long` integers changed

- x86: `int`: 32bits, `long`: 32bits
- x86_64: `int`: 32bits, `long`: 64bits

This causes errors with Fortran interfaces

- Quick and dirty fix: convert all `long`s to `int`s
  - We have a script to do this and handled *most* edge cases
  - A couple of system calls require `long` — but will be documented
  - For network code, consider using C99 fixed size types from `<stdint.h>`
- Also, need some Fortran compiler flags