WELCOME!

Northern Virginia Linux Users Group

Mailing List: http://firemountain.net/mailman/listinfo/novalug

We Meet Every 2nd Saturday of the Month

Meetup: https://www.meetup.com/novalug/

Thank You Ridgeline for sponsoring our meetings!

https://www.ridgelineintl.com/

Everyone is welcome!

INTRODUCTIO
N TO LINUX

By Peter Larsen
For NOVALUG, June 2019
@egoalter, ego.alter@gmail.com

It doesn't have to be like this

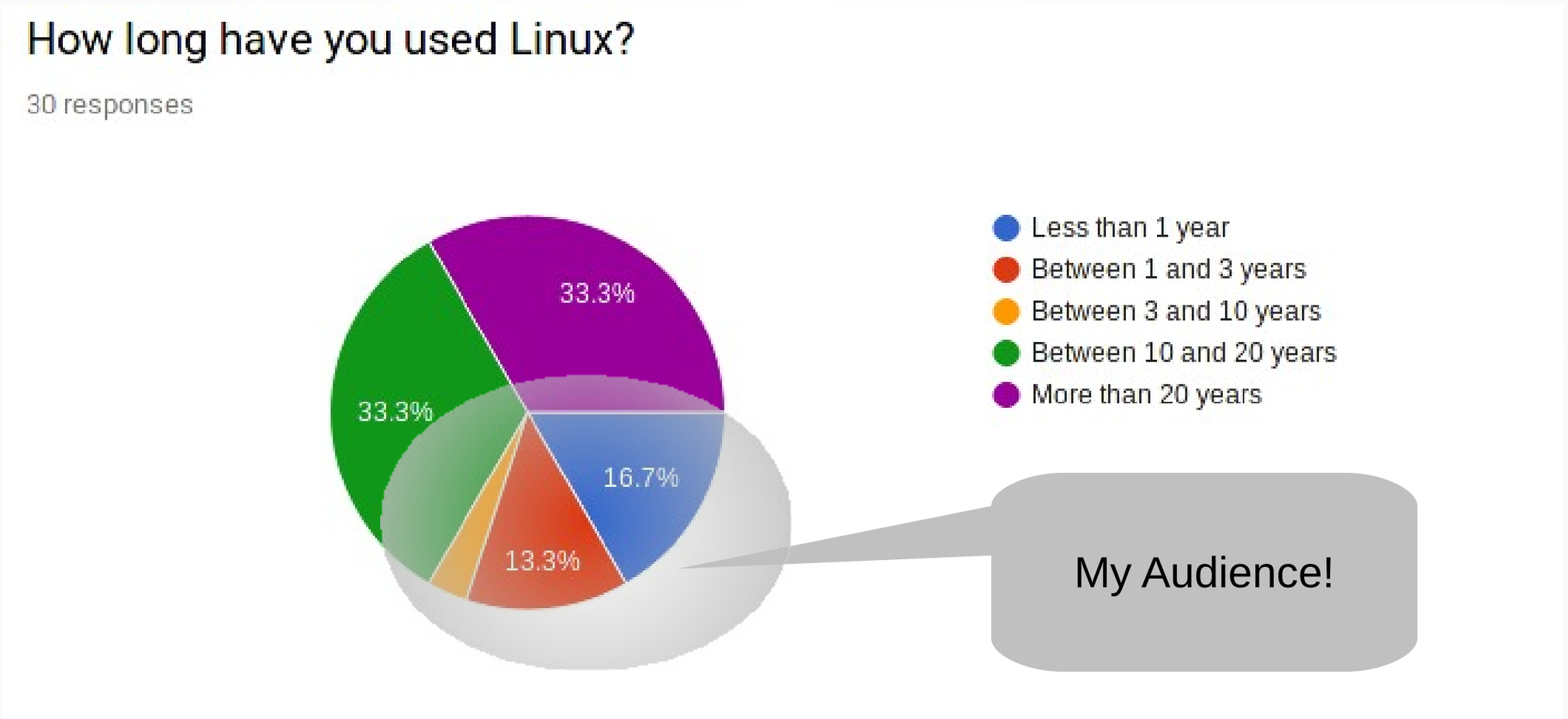
agenda

- Who/What/Why ....
- So what's an Operating System and why should I care?
- How – Getting it/Installing it
- What the ....? What happened? The boot process!
- Let me Service You!
- What are doing? Diagnostics. Logs and getting answers
- Talk to me! Networking
- Tell me again – Bash Scripting
- Long distance relationships – SSH and it's pals
- Thank you

WHO/WHAT/WHY ....

- Who am I?
  - Hacker, Husband, Computer Guy, Linux fan-boy, Open Source evangelist, Retro Computing/Electronics fan, Micro-controllers, Single Board Computers, IoT, 3D Printing – and cussing .... lots of cussing ....
  - Member of NoVALUG since early 2000
- What am I doing here?
  - Worked with Linux since 0.99b (anno 1993). Work with Linux today (Red Hat). Main computing platform since very early 2000. Sole platform since 2006.
  - I explain technology for a living. Making complex things easy to understand is my goal.
- Why are *you* here?
  - Hopefully to learn about Linux; how to become more proficient with Linux
  - Even though more than 60% of you have used Linux for more than 10 years!!

YOU'RE NOT MAKING THIS EASY!



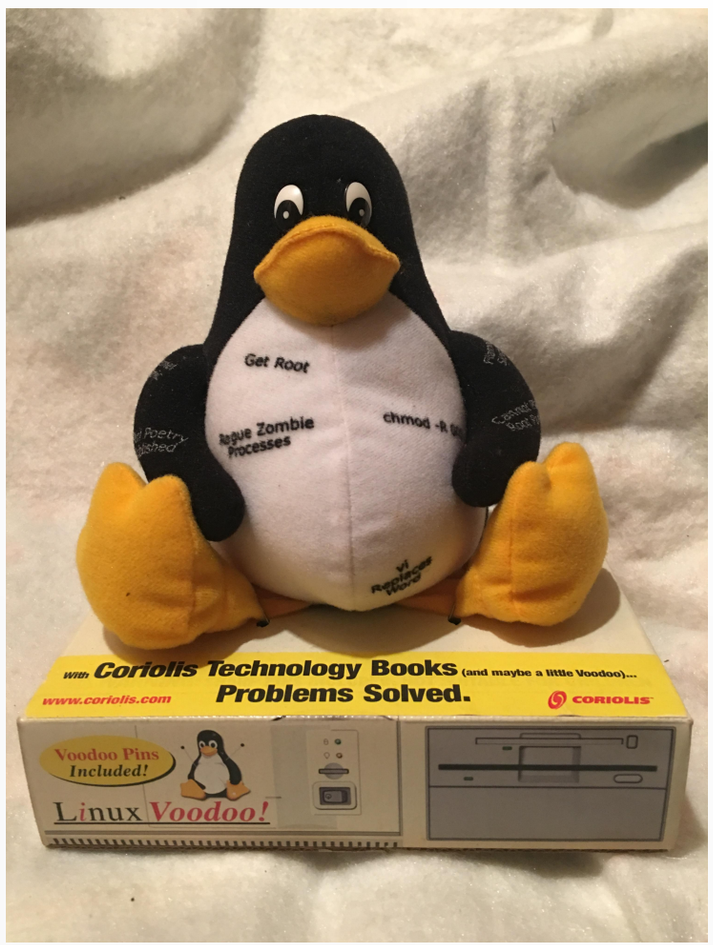
So what's an Operating System and why should I care?

- This isn't your grandfather's Linux
  - Not a necessary evil
  - Makes your hardware blazing fast, easy to manage and allows you to produce cool things
  - It's easy – you don't have to have a PhD to make it work
- Hardware managed by Software => Operating System
- The OS provides
  - Processes, files, printing, security, networking, process isolation, memory management, graphical subsystem
  - Brings your hardware to live

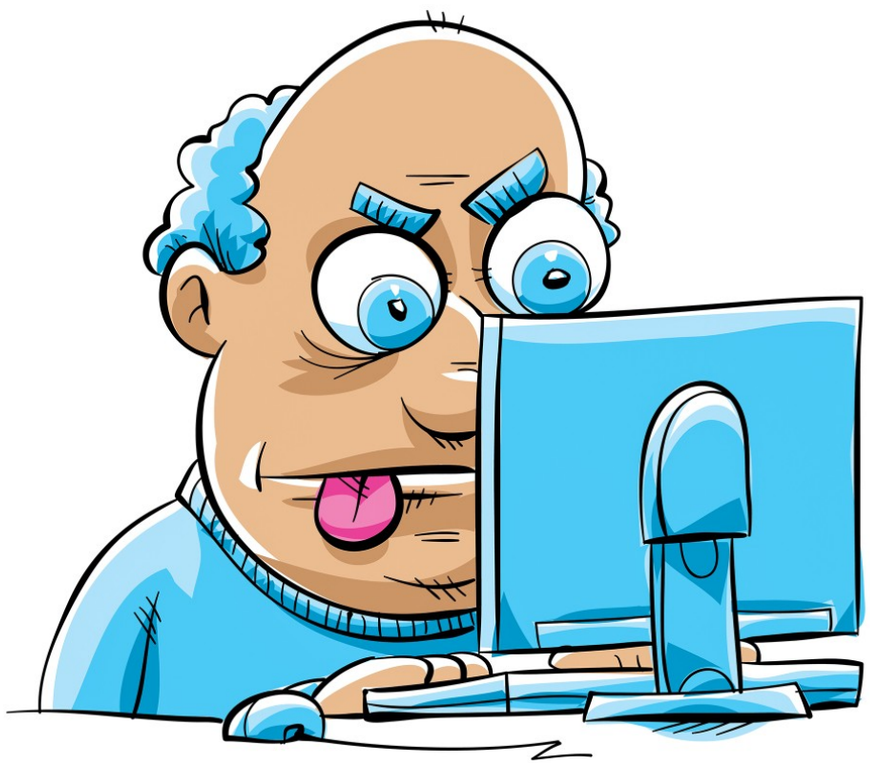
Old Time Slackware Install Guide

- http://www.slackware.com/install/bootdisk.php
- Steps
  - Download right boot disk image for right media
    - IDE
      - Bare, bareacpi, ataraid,lowmem,old_cd,pportide,sata
    - SCSI
      - Adaptec,ibmmca,jfs,raid,scsi,scsi2,scsi3,speakup
  - Download root image
    - Install.1,install.2.install.zip,network.dsk,pcmcia.dsk,rescue.dsk,sbootmgr.dsk
  - Find windows program to write image to floppy
    - Be sure to include parameters for the hardware you'll be installing on
  - Insert media and boot
    - Partition using fdisk (cmd-line), install boot partition, install root partition
    - Command-line paradise!
  - Remove media, boot computer – drink lots of Whiskey hoping it goes well

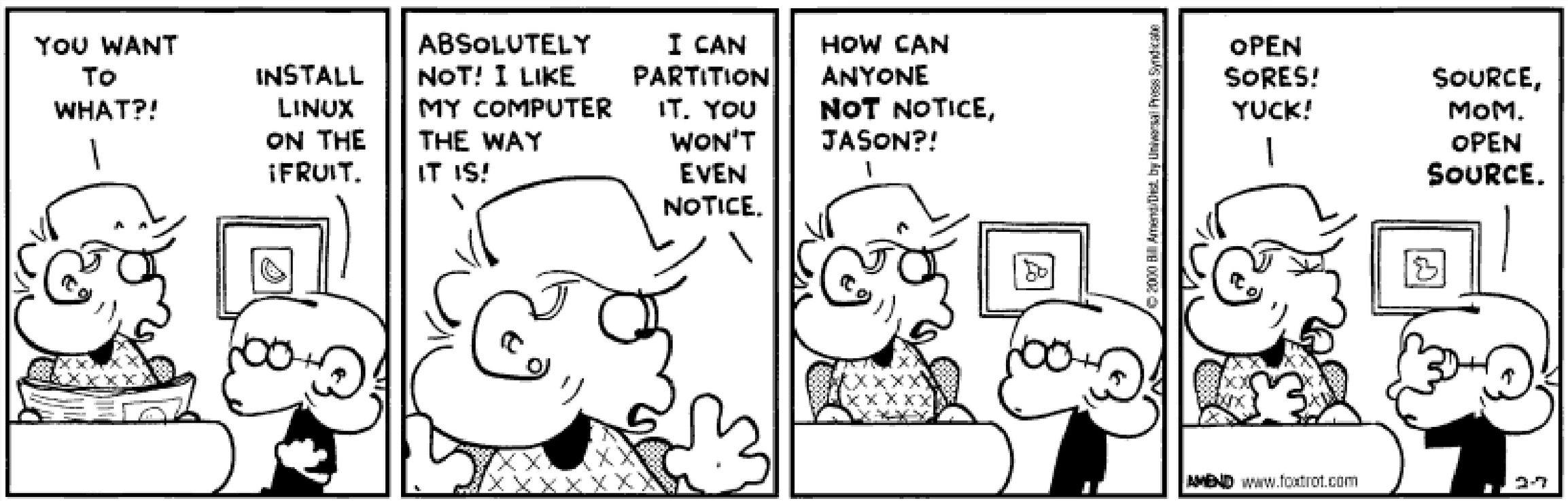
The Voodoo Days are Long Gone



Src: https://imgur.com/gallery/LFoIr



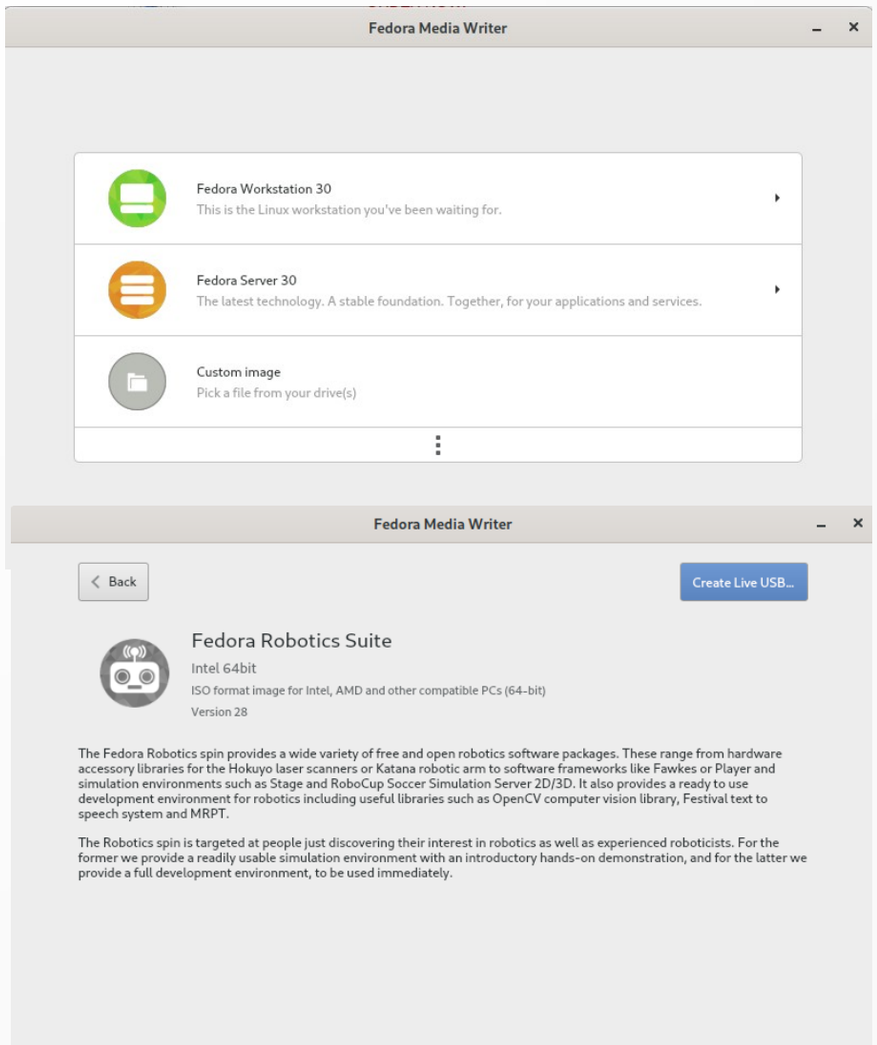
Src: Blambca/Shutterstock



Src: https://aintnogod.com/ipb/gallery/image/2064-linux-cartoon/

Installation Today

- Install “Fedora Media Writer” on your current platform
- Run it – pick your “poison”
- Insert USB
- Sit back and wait .....
- Put USB in any computer
- BOOT
- Fun to be had!!



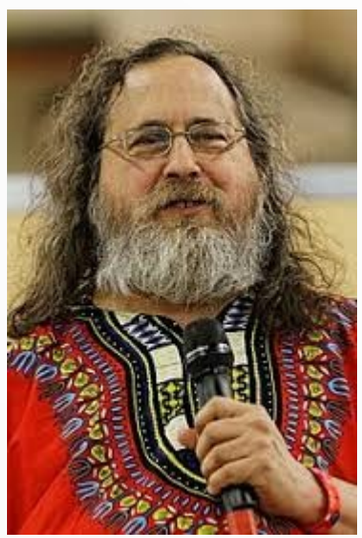
A THANK YOU TO OUR LEGENDS

- Giants who's shoulders we stand on .....

Jon "Maddog" Hall

Hardware getter for Linus!
FSF promoter

Richard Stallman



GPL, GNU, FSF founder

Linus Torvalds

Inventor of Linux
Benevolent Dictator for Life of the Linux kernel

Ken Thompson

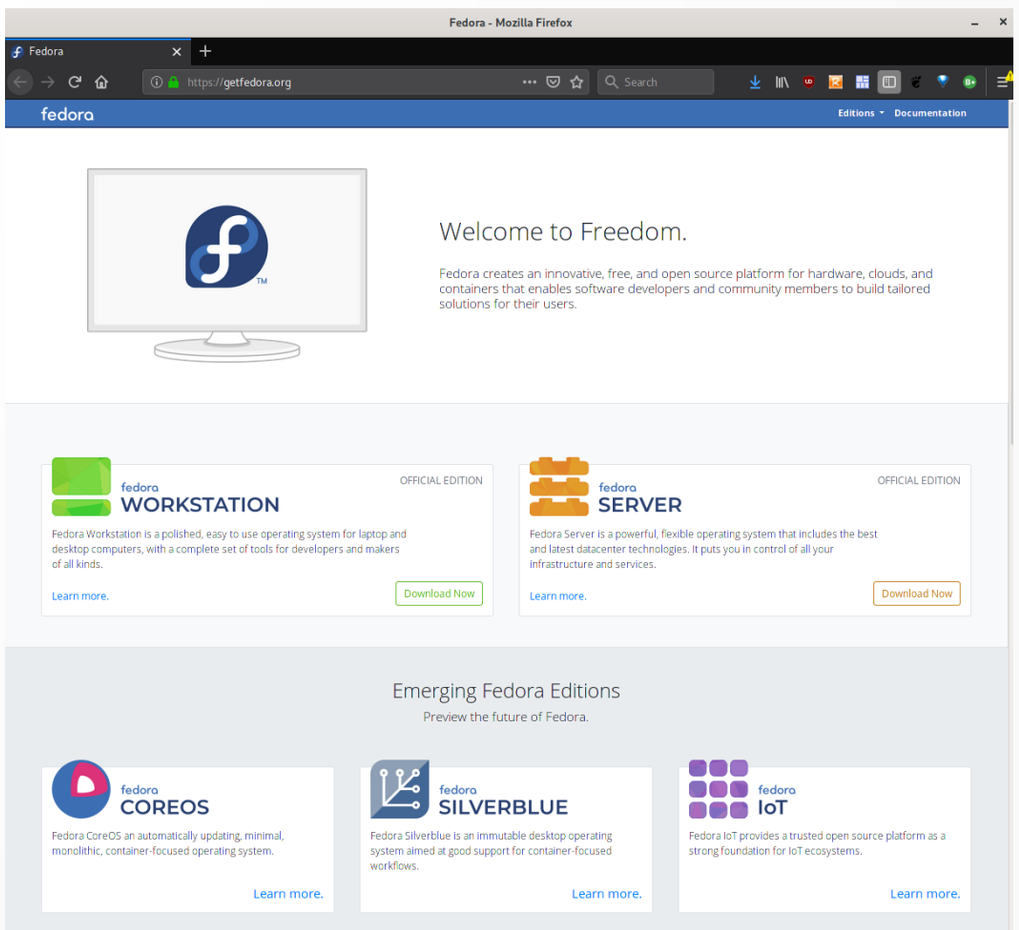
Co-inventor Of Unix

How - Getting it/Installing it

- (Too) Many Choices
- Beginners pick major distros – more than ONE
  - Fedora, SUSE, Ubuntu, CentOS, Arch etc.
- Follow instructions found on your distributions main web page
  - Do not assume instructions for one distro can be applied to another

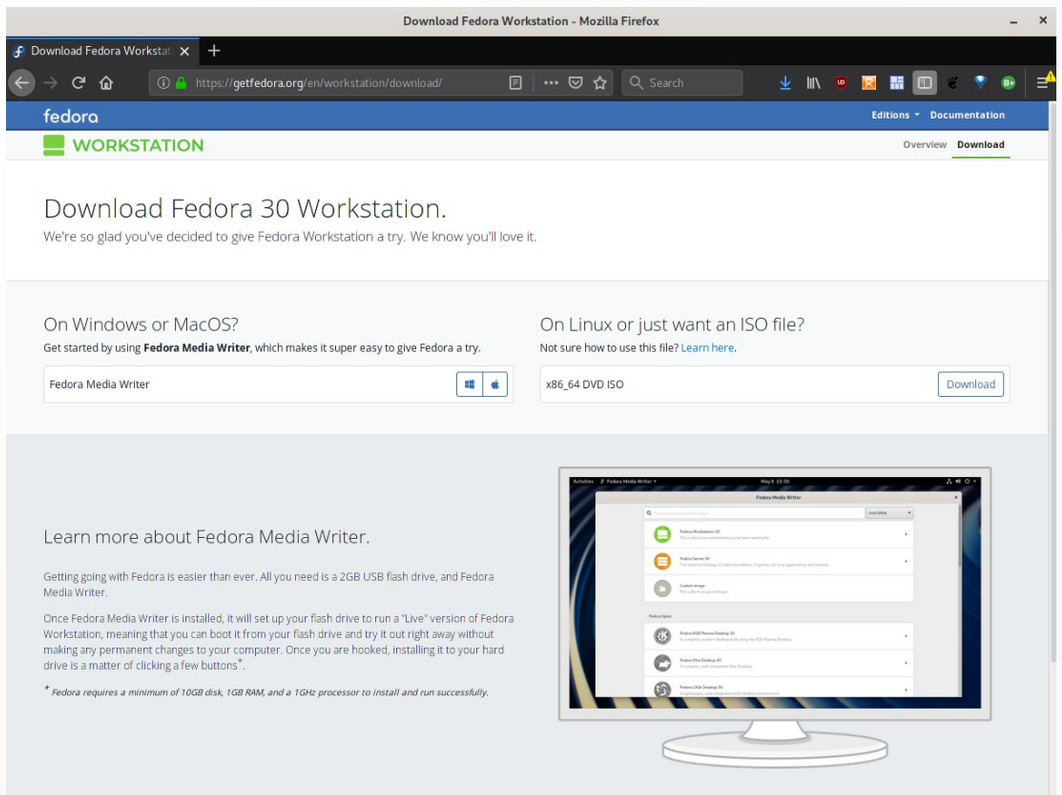
Getting Started with Fedora

- Very active distribution
- 6 month cycle
- Test bed for new technologies – bleeding edge
- Active community – lots of help
- Rock solid
- Fedora = RHEL Upstream



INSTALLATION

- Recommend Media Writer
- ISO is still an option
  - Can be written to USB and DVDROM
- Once Media is written
  - And validated
- Boot on media
- For USB you may need to activate your computers BIOS boot menu (F2/F12/F10)



LET'S TRY IT

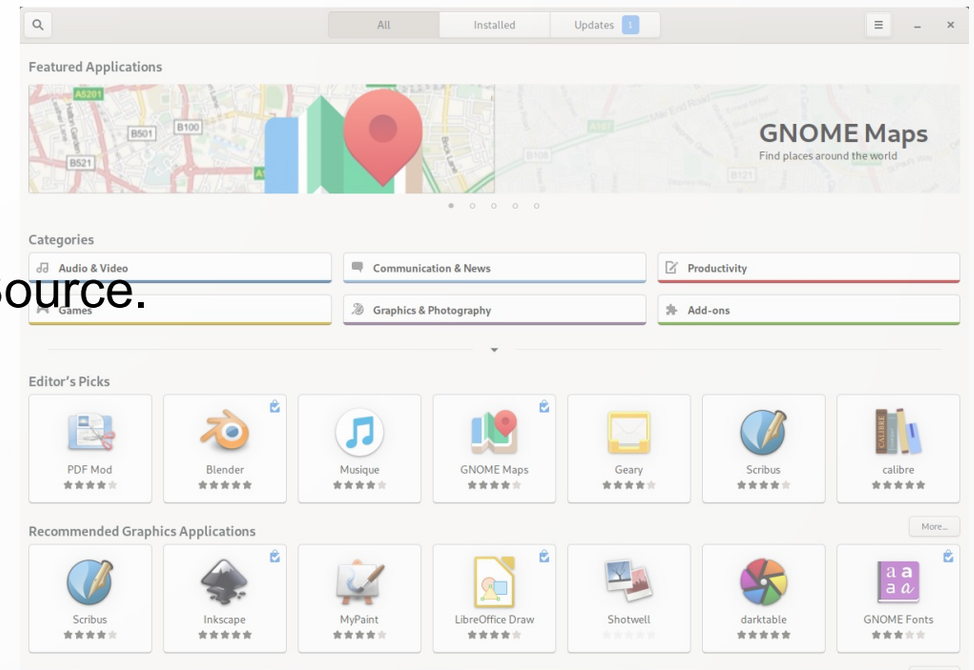
DEMO TIME
Installing Fedora 30 on a Virtual Machine

WHAT THE ....?

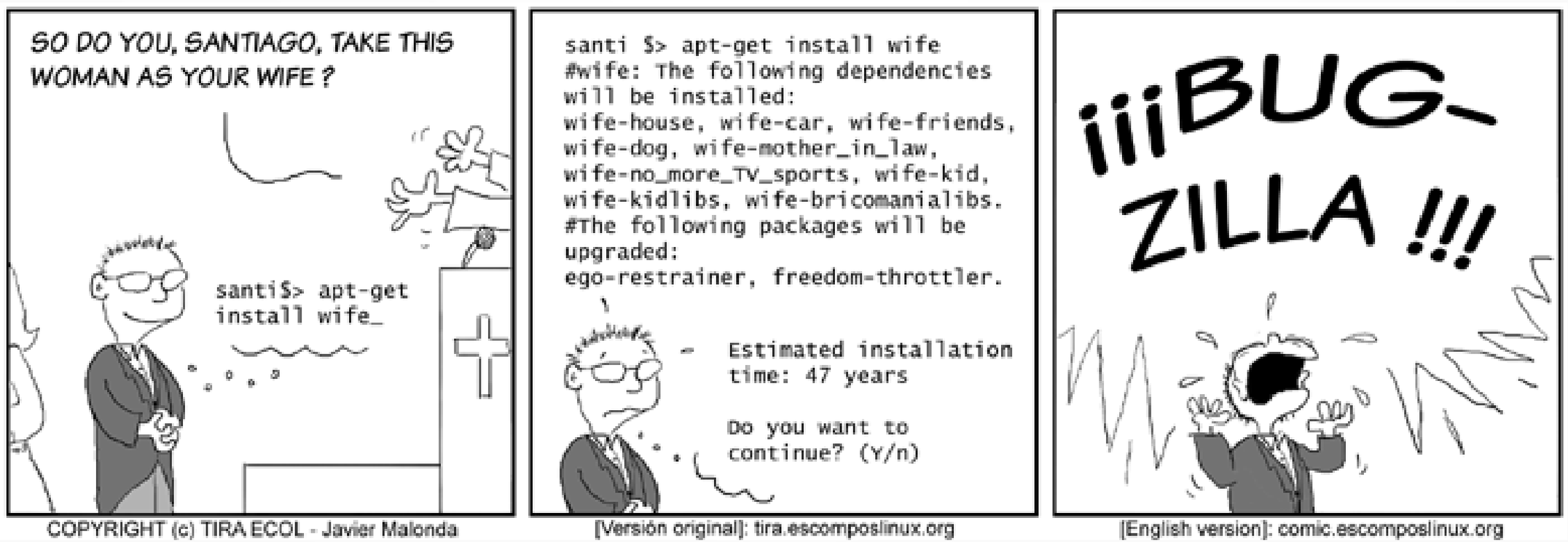
- What happened?
  - The Operating System was installed
  - The Hard Drive (hdd/sdd/nvram) was initialized
  - Boot-loaders was setup – integrated into your BIOS/EFI process
  - Users were created, software was installed (700+ packages!)
  - No techo-talk required
- What Now?
  - You'll want to bring your system up to date
  - Install software you want
  - Start using the computer for the purpose you want

INSTALLING SOFTWARE

- On the main menu/favorite click “Software”
  - Pick your poison – click install
  - All tested/proven with your distribution. All Open Source.
- For the advanced users ....
  - “dnf search <term>”
  - “dnf list <package>”
  - “dnf install <package>”
- Remember to keep your system updated!
  - Message bar on GUI will tell you when updates are available
  - Use “dnf update” to force an update now



Use apt-get on Ubuntu/Debian



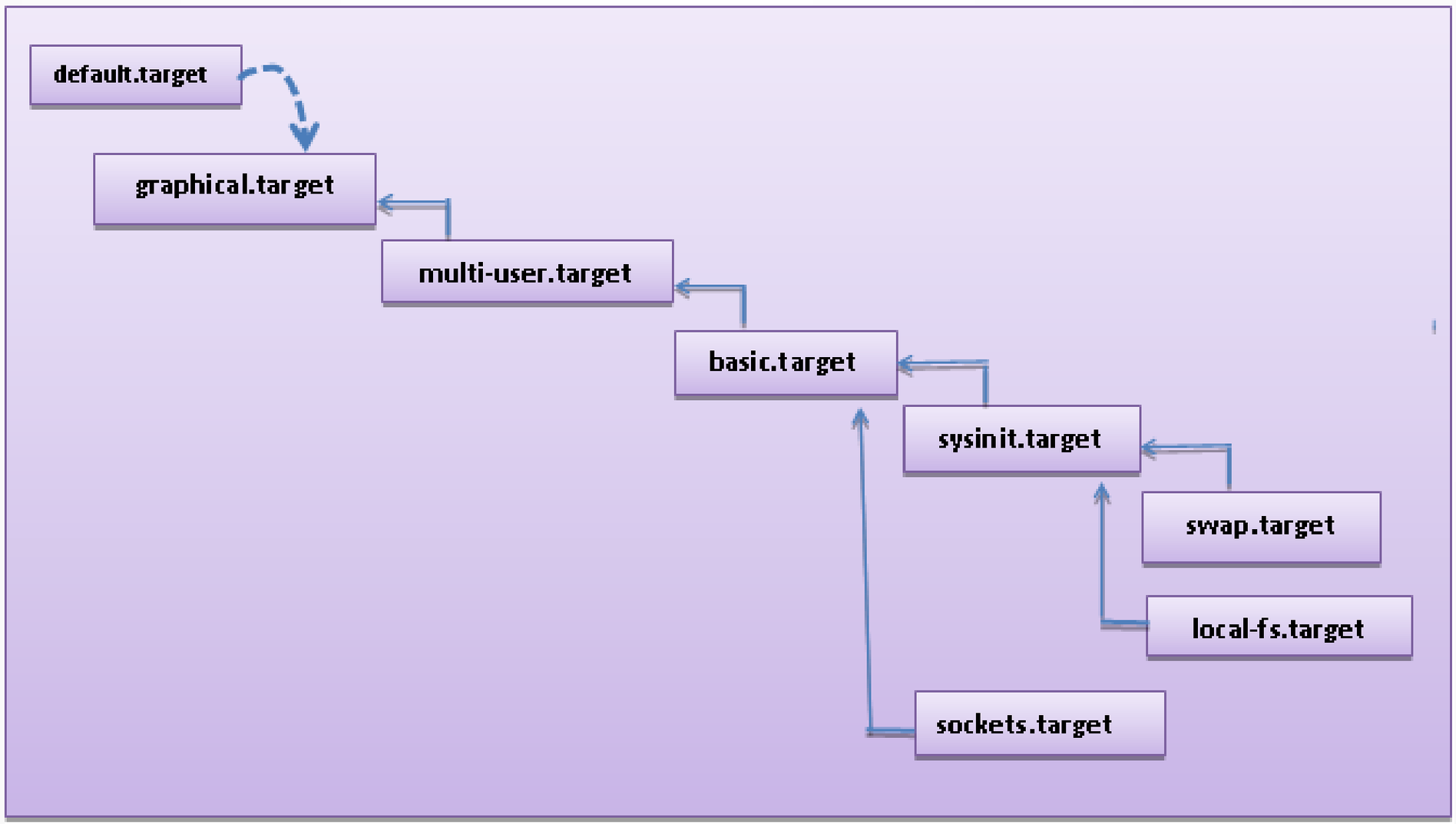
Src: comic.escomposlinux.org

The Boot Process

- Power On
- Power On Self Test (POST)
- UEFI Activated
- UEFI load of Linux boot loader (GRUB)
- Your boot menu shows (maybe)
- Latest kernel loaded
- InitramFS loaded
- System modules initialized

- Root FS loaded/mounted
- System init process (PID 1) started
- System loads services initializes everything for use
- GDM starts – prompts for a login
- Gnome-session process start after successful login
- Loads user set default applications to run
- Sets graphical defaults, background/foreground, colors etc.

SYSTEMD BOOT PROCESS (INIT PROCESS)



Linux and Booting

- GRUB - GNU GRand Unified Bootloader
  - Kernel – the core of Linux. Converts hardware to abstract notions of processes, files, memory etc.
  - InitramFS – Ram disk to boot-strap things
- Root File System (rootfs)
  The only required file system, mounted in /
  Contains software, libraries, settings etc
- Init process (systemd) – PID 1. Owns all processes. Starts everything, controls everything
- Services
  Background processes managing the system resources

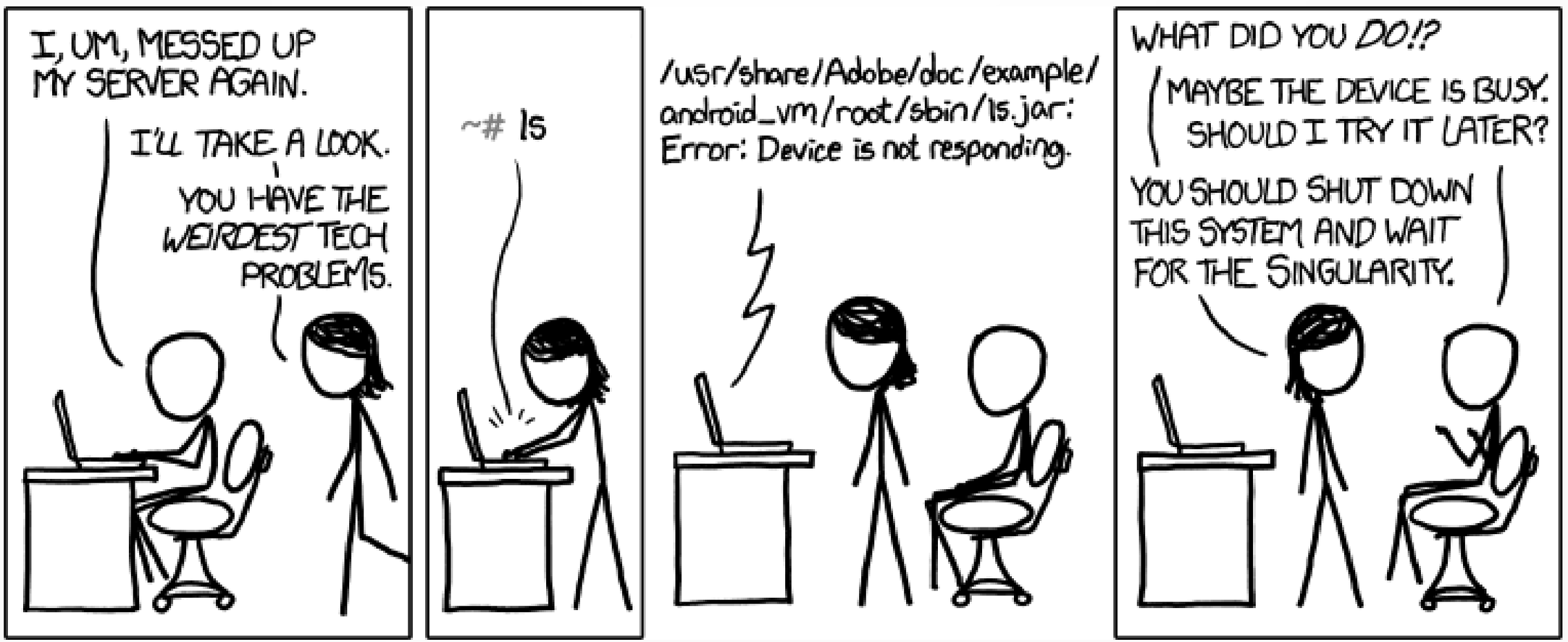
Linux is control

- Boot is complex
- Customizing boot is meant for experienced users
- Everything can be tweaked/modified in Linux – every aspect of the boot process
- Works perfect out of the box for 95% of people
  No customization needed!
- Breaking the boot process will brick your system
- GRUB handles any OS – including Windows
  Dual-boot is handled by GRUB

Some basics

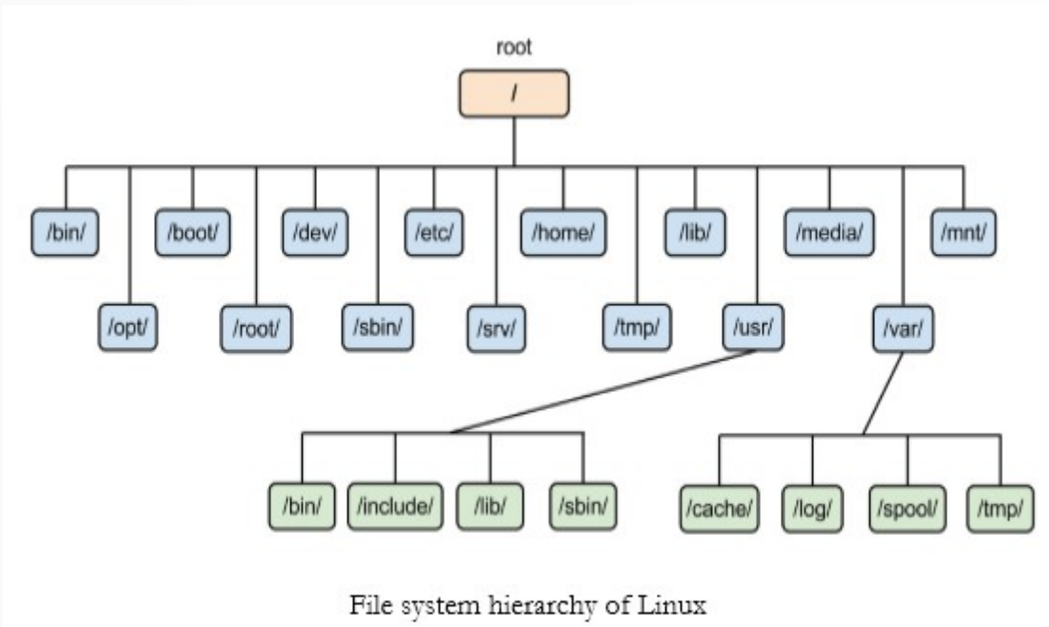
File Systems

The Core of the System



Src: https://xkcd.com/1084/

File system(s)



File system hierarchy of Linux

Src: https://thesagediary.com/2018/09/26/linux-file-system-directory-architecture/

- “everything is a file”
- /proc, /sys, /dev are special file system – not for normal files
- /home is typically where all user's home directories are placed
- /bin is linked to /usr/bin
- /sbin is linked to /usr/sbin

Overview - Directories

- Every user has a home directory
  - /home/peter
- There's always a working directory – default to $HOME
- Directories organize files. Contains:
  - Files
  - Directories
- Separate directories with /
- Directory commands
  - $ cd # change directory
  - $ pwd # print working directory
- Special directories
  - . # dot – current directory
  - .. # two dots – previous/parent directory
- Access to a directory, only if you have access to ALL parent directories

Overview - Files

- A directory is just a special kind of file (everything is a file)
- Files belong to a directory
- Filename must be unique
- Special characters can be part of filenames using “ “
  - Except /
- File commands
  - $ ls # list directory content (list files)
  - $ cat # print content of file
  - $ file # print type of file
  - $ chmod # change security of files
- Links – symlink/hardlink

File Systems - The Quick Guide

- A file system is a structure put onto a block device
- A file system must be mounted to the “tree” to be active
- Lots of different file systems exist:
  - vfat, ext2, ext4, ntfs, brtfs, xfs, reiserfs
- You can mount network based file systems too (like nfs)
- Only root can mount disks
  - Exception - “fuse” allows users to mount under some circumstances, such as your USB when you insert it

Special File Systems

- /proc
  - Kernel configuration/state files
- /sys
  - Kernel modules current settings
- /dev
  - Device definitions (mknod – special files)
- /boot
  - A real file system, but special structure needed for boot

Let me Service You

Systemd and system processes

Let me service you!

- Introducing “systemd” – no cussing now!
- Controls what runs, where and how on your system
- Some terms (but not all)
  - **Service**: A process with a specific purpose. Ie. httpd.service is the Apache web server
  - **Target**: Defines a group of services (and more) to be activated. Once activated the target is considered reached
    - graphical.target, multi-user.target, recovery.target ....
  - **Socket**: When activity on a socket happens (network or otherwise) this process runs
  - Mount: A mount point, mounts can be static or automated
- Controlled/managed via “systemctl” command

systemctl

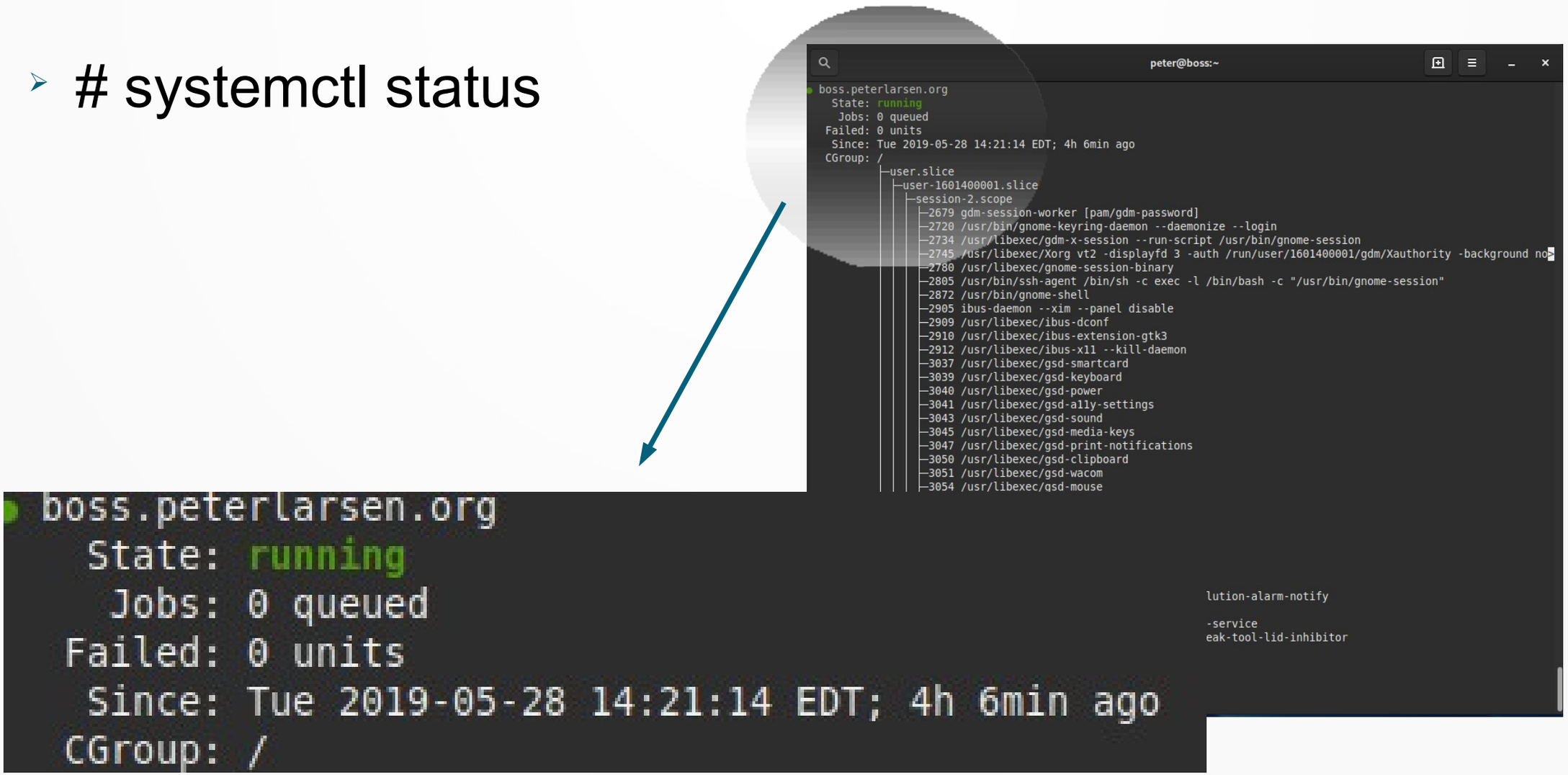
- Can be run with or without root. Run as a user, it can only manage/show user processes.
- # systemctl status
- # systemctl status httpd.service
- # systemctl start
- # systemctl --failed
- # systemctl stop
- # systemctl isolate <target>
- # systemctl -t service|target|mount|socket

SYSTEMCTL BOOT TARGETS

- Important targets (used to be run-levels):
  - graphical.target
  - multi-user.target
  - emergency.target
- Disable graphical environment
  - $ systemctl isolate multi-user.target
- Enable graphical environment
  - $ systemctl isolate graphical.target
- Current boot target
  - $ systemctl get-default
- New boot target
  - $ systemctl set-default multi-user.target

HOW'S MY SYSTEM DOING?

- # systemctl status



Web Server Management

- Is there a service?
  - systemctl status httpd.service
- Yes!
  - Disabled
  - Not Running

```
[peter@boss ~]$ systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
  Drop-In: /usr/lib/systemd/system/httpd.service.d
           └─php-fpm.conf
   Active: inactive (dead)
     Docs: man:httpd.service(8)
```

- # systemctl start httpd
  - Returns nothing – this is linux. It means “all ok”
- # systemctl status httpd

Running Service Status

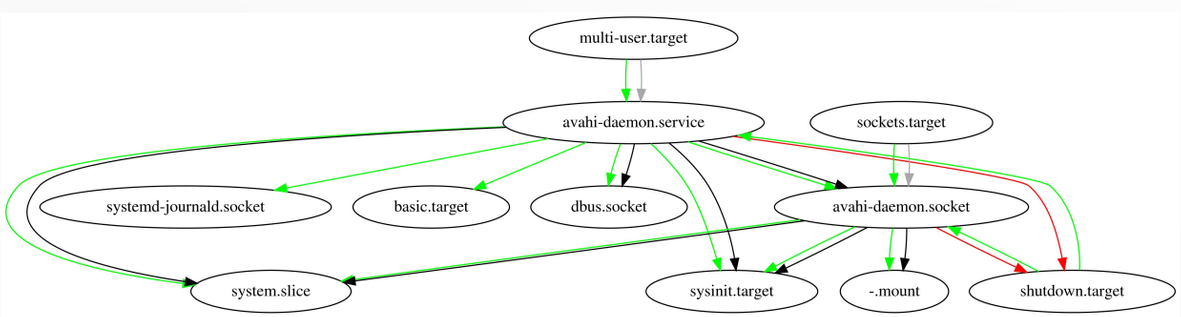
```
[peter@boss ~]$ systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
  Drop-In: /usr/lib/systemd/system/httpd.service.d
           └─php-fpm.conf
   Active: active (running) since Tue 2019-05-28 18:37:10 EDT; 43s ago
     Docs: man:httpd.service(8)
 Main PID: 9650 (httpd)
   Status: "Running, listening on: port 80"
    Tasks: 213 (limit: 4915)
   Memory: 21.1M
   CGroup: /system.slice/httpd.service
           ├─9650 /usr/sbin/httpd -DFOREGROUND
           ├─9657 /usr/sbin/httpd -DFOREGROUND
           ├─9659 /usr/sbin/httpd -DFOREGROUND
           ├─9660 /usr/sbin/httpd -DFOREGROUND
           └─9661 /usr/sbin/httpd -DFOREGROUND

May 28 18:37:09 boss.peterlarsen.org systemd[1]: Starting The Apache HTTP Server...
May 28 18:37:10 boss.peterlarsen.org httpd[9650]: Server configured, listening on: port 80
May 28 18:37:10 boss.peterlarsen.org systemd[1]: Started The Apache HTTP Server.
```

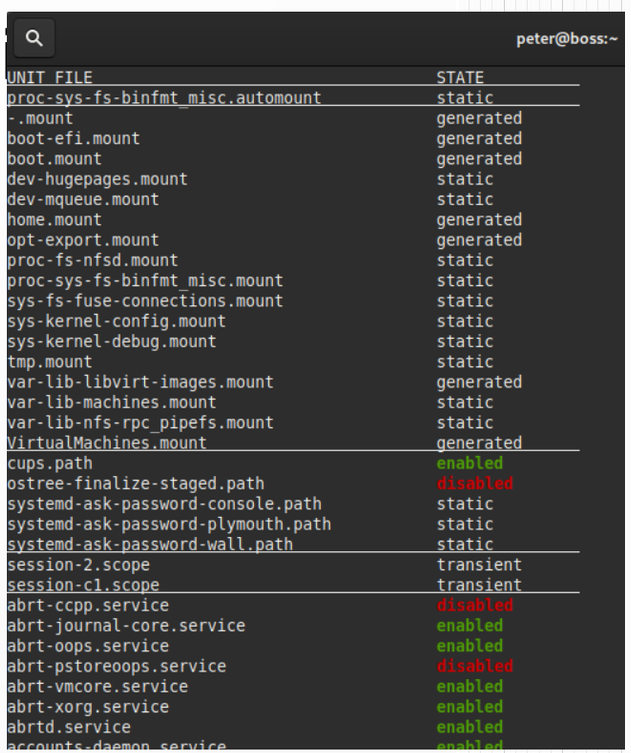
How to run a service on boot

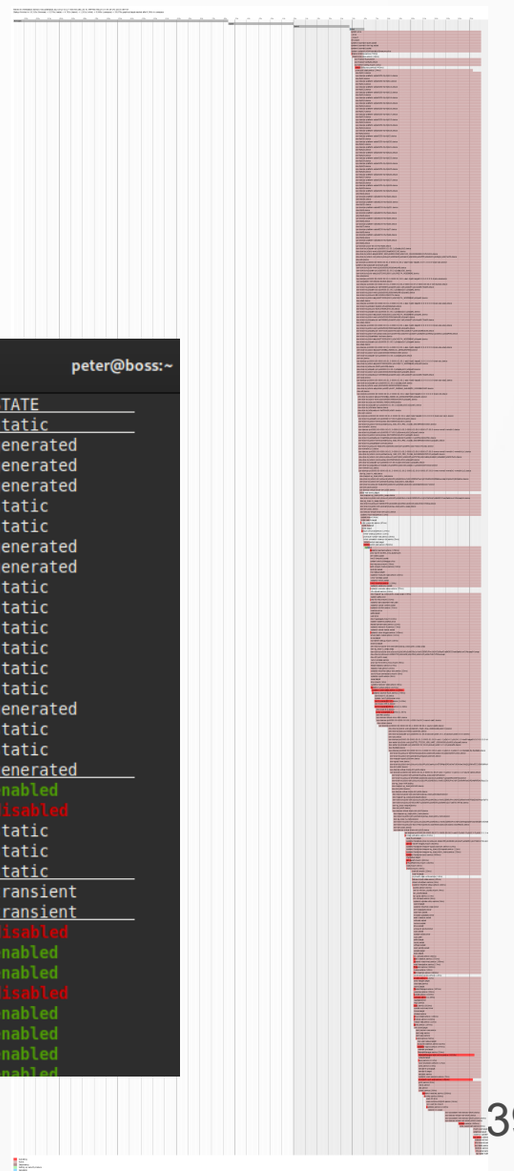
- Service must be enabled to be activated at boot
- # systemctl enable httpd.service
- To start and enable a service with the same command
- # systemctl enable --now httpd.service
- Note – for services the “.service” is optional

Advanced systemd commands

- # systemd-analyze
  - Blame – what's taking so long at boot?
  - Plot – Describe all tasks and tim graphically
  - Dot – dependency graph







SYSTEMD UNIT DETAILS

- # systemctl show <unit>

```
$ systemctl show httpd.service | grep Exec
ExecMainStartTimestamp=Tue 2019-05-28 18:37:09 EDT
ExecStart={ path=/usr/sbin/httpd ; argv[]=/usr/sbin/httpd $OPTIONS
-DFOREGROUND ; ignore_errors=no ; start_time=[Tue 2019-05-28 18:37:09
EDT] ; stop_time=[n/a] ; pid=9650 ; code=(null) ; status=0/0 }
ExecReload={ path=/usr/sbin/httpd ; argv[]=/usr/sbin/httpd $OPTIONS -k
graceful ; ignore_errors=no ; start_time=[n/a] ; stop_time=[n/a] ; pid=0 ;
code=(null) ; status=0/0 }
```

- Custom service files in */etc/systemd/user/*
- System defined service files in */usr/lib/systemd/system/*
- You can customize system units in */etc/systemd/system/*

What's going on?

Diagnostics and logs

Getting Answers

- Did you notice the log output from systemctl status??
- Journald is responsible for all log data from everything running
- No longer will there be messy files in */var/log/*
  By default journald keeps the logs in binary searchable databases (journal files).
- Read journald data using “journalctl” command.
- Very flexible. Searches by unit, message type, time, boot etc. etc.

I WANT TO SEE MY "MESSAGE" LOG!

- $ journalctl

- Yup – that's it
- You typed more when you wrote:
  - $ sudo tail -100 /var/log/messages

BASIC JOURNALCTL USAGE

- **$ journalctl -u avahi-daemon -b 0**
- Show logs since this computer started for the service “avahi-daemon”

```
[peter@boss systemd]$ journalctl -u avahi-daemon -b 0
-- Logs begin at Tue 2019-04-16 09:44:02 EDT, end at Tue 2019-05-28 19:52:12 EDT. --
May 28 18:21:19 boss.peterlarsen.org systemd[1]: Starting Avahi mDNS/DNS-SD Stack...
May 28 18:21:19 boss.peterlarsen.org avahi-daemon[1602]: Found user 'avahi' (UID 70) and group 'avahi' (GID 70).
May 28 18:21:19 boss.peterlarsen.org avahi-daemon[1602]: Successfully dropped root privileges.
May 28 18:21:19 boss.peterlarsen.org avahi-daemon[1602]: avahi-daemon 0.7 starting up.
May 28 18:21:19 boss.peterlarsen.org avahi-daemon[1602]: Successfully called chroot().
May 28 18:21:19 boss.peterlarsen.org avahi-daemon[1602]: Successfully dropped remaining capabilities.
May 28 18:21:19 boss.peterlarsen.org avahi-daemon[1602]: No service file found in /etc/avahi/services.
May 28 18:21:19 boss.peterlarsen.org avahi-daemon[1602]: Network interface enumeration completed.
May 28 18:21:19 boss.peterlarsen.org avahi-daemon[1602]: Server startup complete. Host name is boss.local. Local service cookie is 2137494883.
May 28 18:21:19 boss.peterlarsen.org systemd[1]: Started Avahi mDNS/DNS-SD Stack.
May 28 18:21:21 boss.peterlarsen.org avahi-daemon[1602]: Joining mDNS multicast group on interface virbr0.IPv4 with address 192.168.122.1.
May 28 18:21:21 boss.peterlarsen.org avahi-daemon[1602]: New relevant interface virbr0.IPv4 for mDNS.
May 28 18:21:21 boss.peterlarsen.org avahi-daemon[1602]: Registering new address record for 192.168.122.1 on virbr0.IPv4.
May 28 18:21:24 boss.peterlarsen.org avahi-daemon[1602]: Joining mDNS multicast group on interface enp6s0.IPv6 with address fe80::13e2:4d4c:b1cd:5579.
May 28 18:21:24 boss.peterlarsen.org avahi-daemon[1602]: New relevant interface enp6s0.IPv6 for mDNS.
May 28 18:21:24 boss.peterlarsen.org avahi-daemon[1602]: Registering new address record for fe80::13e2:4d4c:b1cd:5579 on enp6s0.*.
May 28 18:21:24 boss.peterlarsen.org avahi-daemon[1602]: Joining mDNS multicast group on interface enp6s0.IPv4 with address 192.168.12.15.
May 28 18:21:24 boss.peterlarsen.org avahi-daemon[1602]: New relevant interface enp6s0.IPv4 for mDNS.
May 28 18:21:24 boss.peterlarsen.org avahi-daemon[1602]: Registering new address record for 192.168.12.15 on enp6s0.IPv4.
```

Advanced Log Viewing

- $ journalctl --utc
  Show all timestamps in UTC (timezone)
- $ journalctl –list-boots

```
-5 b1a90739e314461889862caad66fd1c9 Sun 2019-05-26 06:54:31 EDT—Sun 2019-05-26 19:02:40 EDT
-4 4db50cd0c2a7410cbf9d066759f90783 Mon 2019-05-27 08:11:17 EDT—Mon 2019-05-27 22:11:33 EDT
-3 75088d61279a4083b6fe8e1f0405362e Tue 2019-05-28 06:44:48 EDT—Tue 2019-05-28 15:57:44 EDT
-2 5c1999b5be4d44b797577391be276d4f Tue 2019-05-28 11:58:25 EDT—Tue 2019-05-28 15:59:46 EDT
-1 cb195d8d55854119adc2633964df7ba1 Tue 2019-05-28 12:00:22 EDT—Tue 2019-05-28 18:20:42 EDT
 0 a989c68c0ff24a868baba117eec7f61a Tue 2019-05-28 14:21:13 EDT—Tue 2019-05-28 20:01:05 EDT
```

- $ journalctl -b -1
  Show logs from the previous boot
- $ journalctl --since=<time> --until=<time>
  - journalctl --since 09:00 --until “2 hours ago”
- $ journalctl _PID=<pid>
- $ journalctl -k
  Kernel messages – also known as **dmesg** if you just want the current kernel messages.

EVERYTHING PRODUCES LOGS

- The kernel when loading is very chatty
  - $ dmesg (or journalctl --dmesg or journalctl -k)
  - Show all kernel messages since boot
  - Hardware and system issues are typically found here
- Sometimes you have to turn up verbosity to see what's really going on
- “Does not work” isn't a diagnosis!
- Look for messages that contradict expected functionality

```
#
# ErrorLog: The location of the error log file.
# If you do not specify an ErrorLog directive within a <VirtualHost>
# container, error messages relating to that virtual host will be
# logged here.  If you *do* define an error logfile for a <VirtualHost>
# container, that host's errors will be logged there and not here.
#
ErrorLog "logs/error_log"

#
# LogLevel: Control the number of messages logged to the error_log.
# Possible values include: debug, info, notice, warn, error, crit,
# alert, emerg.
#
LogLevel warn
```

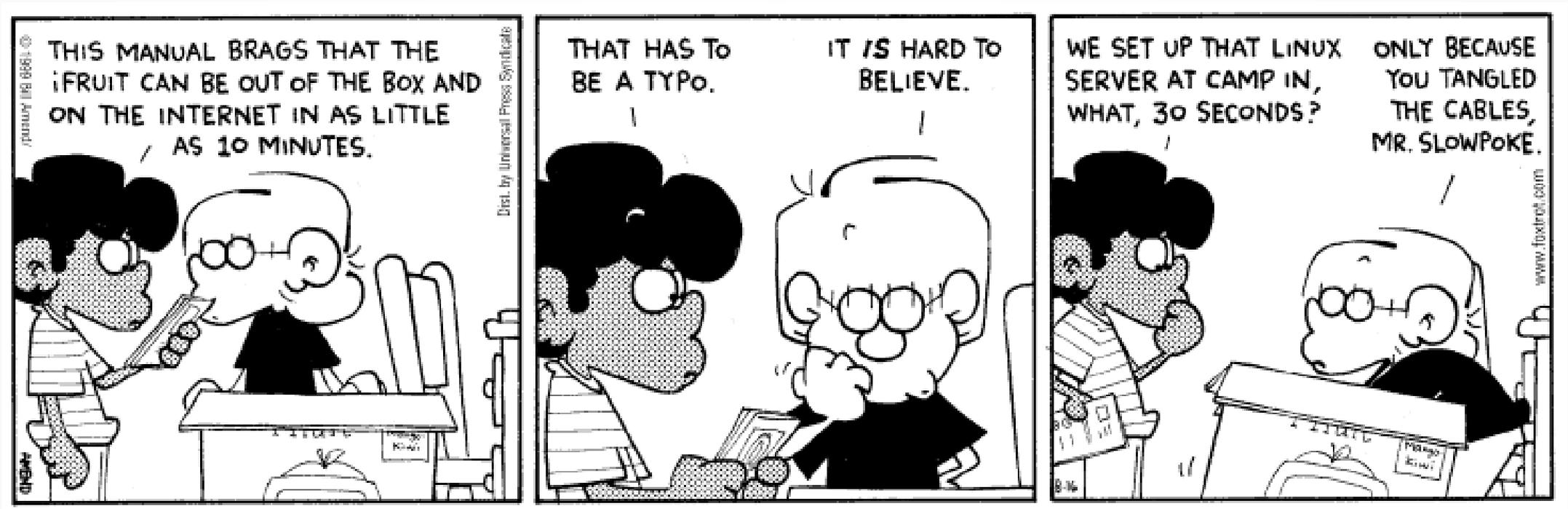
File/Report Bugs!

- If you encounter things that do not work:
  - Try to reproduce the issue on another machine if possible
  - Update to latest version
  - If still an issue, create a bugzilla/jira etc.
    Projects use different reporting tools – so look at the project web-site for a reporting tool.
  - Major BZ site for Fedora
    https://bugzilla.redhat.com/
  - Provide a short description of issue, expected results and what you're receiving. Enclose all the log data you can find
- “abrt” will automatically report crashes if you enable it

Talk to me!

NETWORKING

IT'S EASY!

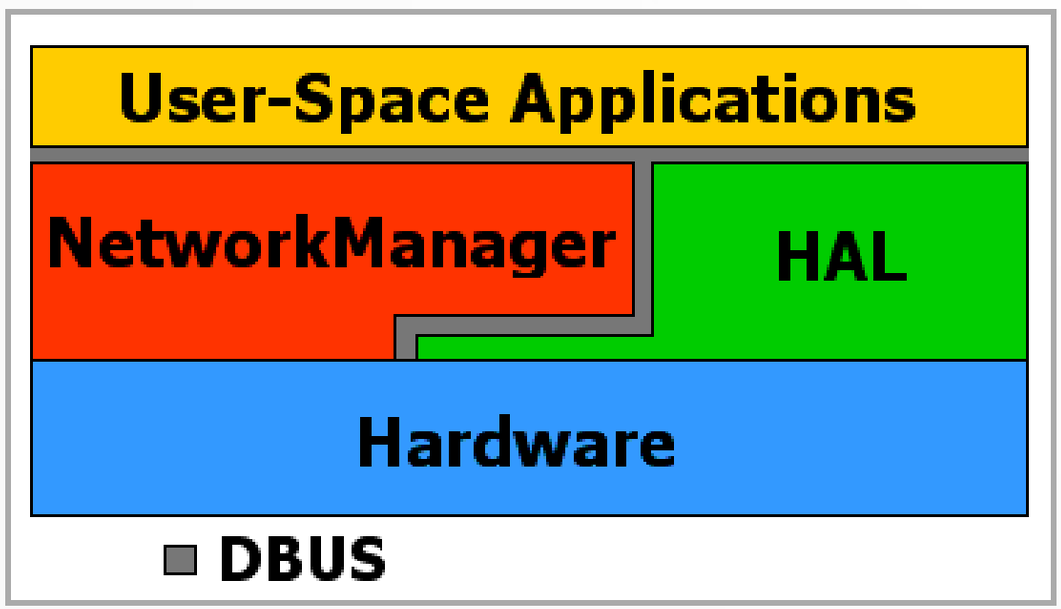


Src: http://folk.uio.no/hpv/linuxtoons/foxtrot.1999-08-16.png

Network Manager

- Network Manager aka NetworkManager
  - See, no acronym!!
- Network Manager is a single place to manage devices, connections, VPN, Bridges, WiFi and other radios
- Network Manager is integrated into dbus and systemd. It reacts automatically to changes and reports changes back, so services can adjust as network availability changes
- Nice GUI built into Gnome, KDE and many other DE's to easily setup WiFi and other networks

What can Network Manager do?

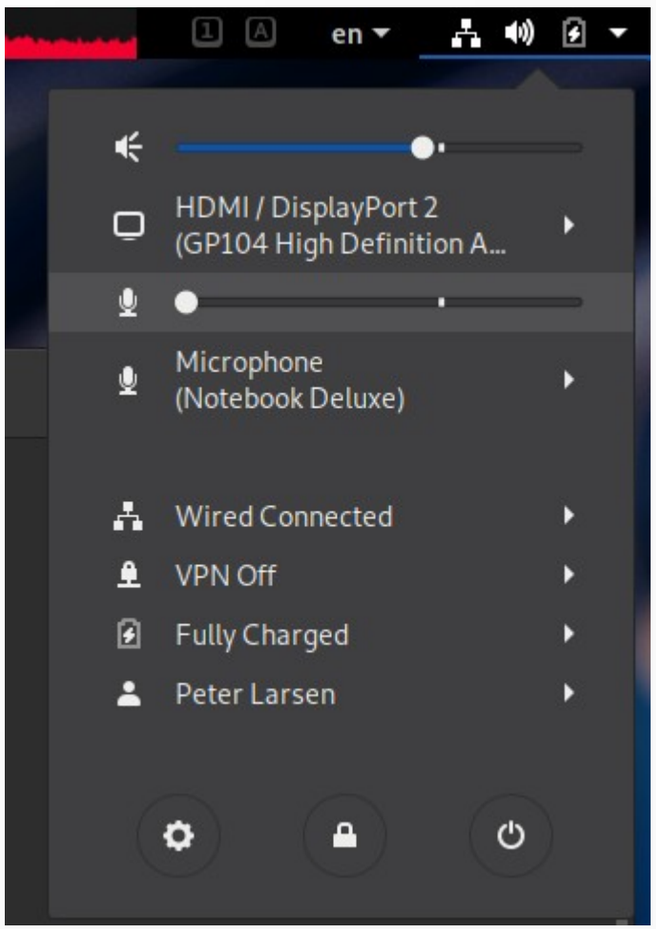


Src: https://www.linuxjournal.com/article/7745

- Notifies the system about network changes
- Manage
  - Wire Network
  - WiFi
  - Bluetooth
  - VPN
  - Mobile Network

Network Management

- Note we did not configure networking when installing but it worked right away?
- Ordinary use do not require any network setup. Your computer gets an IP address and everything is happy
- Gnome and other displays are Network Manager components



BASIC COMMANDS

- $ **nmcli**
  General information of all networking stuff.
- $ **nmcli c**
  Show all connections
- $ **nmcli d**
  Show all network devices
- $ **nmcli r**
  Show all radio device status

```
[peter@boss conf]$ nmcli
enp6s0: connected to enp3s0
        "Intel I211"
        ethernet (igb), 4C:ED:FB:73:BD:8F, hw, mtu 1500
        ip4 default
        inet4 192.168.12.15/24
        route4 192.168.12.0/24
        route4 0.0.0.0/0
        inet6 fe80::13e2:4d4c:b1cd:5579/64
        route6 fe80::/64
        route6 ff00::/8

virbr0: connected to virbr0
        "virbr0"
        bridge, 52:54:00:09:47:BE, sw, mtu 1500
        inet4 192.168.122.1/24
        route4 192.168.122.0/24

lo: unmanaged
        "lo"
        loopback (unknown), 00:00:00:00:00:00, sw, mtu 65536

virbr0-nic: unmanaged
        "virbr0-nic"
        tun, 52:54:00:09:47:BE, sw, mtu 1500

DNS configuration:
        servers: 192.168.11.12 192.168.11.45
        interface: enp6s0
```

Connections?

- A device represents a network device in system
- A connection is a particular configuration which is applied to a device
- Multiple connections for a device
  - Ie. multiple WiFi definitions
  - Work/home network settings
- Advanced features also supported
  - Teaming, QoS, VPN, Virtual Networks etc.

Setting up network from nmcli

- Required knowledge for any administrator certification exam.
- Required when setting up head-less servers – or servers in the cloud
- A LOT easier than creating a random text file and setting variables that if spelled wrong are ignored

STATIC IP SETUP EXAMPLE

- Added a new network device to the demo install
- # nmcli d

```
[root@localhost ~]# nmcli d
DEVICE  TYPE      STATE                                  CONNECTION
enp1s0  ethernet  connected                              Wired connection 1
enp7s0  ethernet  connecting (getting IP configuration)  Wired connection 2
lo      loopback  unmanaged                              --
```

- Note the device names: enp1s0 and enp7s0 – these aren't “eth0” and “eth1”
- Predictable Names – we name devices so we can easily find them in the hardware. “eth0” tells you nothing about what port in the back of the customer with 8 ethernet ports
  - “en” -> EtherNet
  - “p1” -> Bus/port number (1)
  - “s0” -> Slot number (0)

Get on with it - Create network

- A new device was created
- But no IP!

```
[root@localhost ~]# nmcli d
DEVICE  TYPE      STATE                                  CONNECTION
enp1s0  ethernet  connected                              Wired connection 1
enp7s0  ethernet  connecting (getting IP configuration)  Wired connection 2
lo      loopback  unmanaged                              --
```

```
[root@localhost ~]# ip a show enp7s0
3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen
    link/ether 52:54:00:cb:11:02 brd ff:ff:ff:ff:ff:ff
[root@localhost ~]#
```

- Connection was created
  but not associated with a device
- # nmcli c up "Wired connection 2" ifname enp7s0
  - It times out?? What happened?
  - # journalctl -xe NM_CONNECTION=feb882a4-b884-348f

```
[root@localhost ~]# nmcli c
NAME                UUID                                  TYPE      DEVICE
Wired connection 1  876052c7-db7a-33d5-a832-cdccb3016a52  ethernet  enp1s0
Wired connection 2  feb882a4-b884-348f-84fd-2bc931681523  ethernet  --
[root@localhost ~]#
```

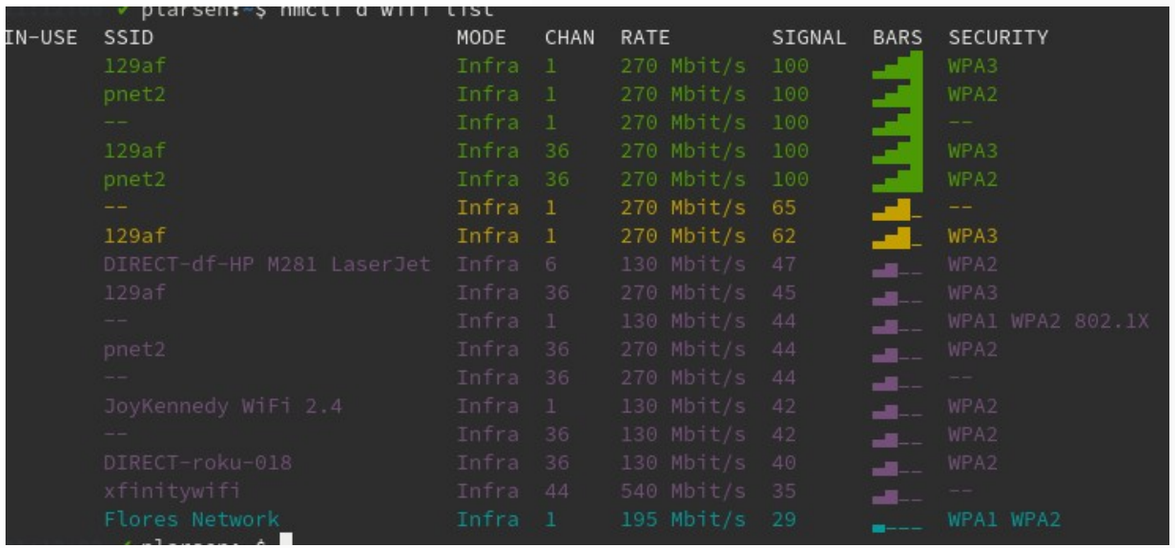
- No DHCP! We have to this totally by hand!

Adding Static IP the Hard Way

- # nmcli c mod “Wired connection 2” ipv4.method manual
- # nmcli c mod "Wired connection 2" ipv4.addresses 192.168.190.5/2”
- # nmcli c mod “Wired connection 2” ipv4.gateway 192.168.190.1
- We have a basic setup working – yay!
- Create our own connection (on one line):
  # nmcli c add con-name mycon ifname enp0s7 type ethernet ipv4 192.168.190.5/24 gw4 192.168.190.1
- But you do not have to remember all the variables!!
  # nmcli c edit “Wired connection 2”
  This will enter an editor where you can see all the settings and just change them interactively.

OTHER COOL NETWORK MANAGER CLI

- # nmcli d wifi list
  Show me some wifi!
- # nmcli d status
  Status of all devices



- Show me all the details
  nmcli -p -m multiline -f all con show

Tell me again ....

BASH Scripting

BASH SCRIPTING

- Bourne Again Shell – BASH
- The defacto standard shell on Linux
  - Plenty of alternatives: ksh, csh, fish, sh ….
- Most distributions still include other shells too
- Bash is the program that runs in the terminal window
  - Reads input from keyboard
  - Outputs to the terminal window
- The shell was one of the initial huge selling points for Unix – extremely powerful and flexible compared to earlier systems
- Shells were created in the times of teletypes – not CRTs!

Command Line Basics

- Note – most things here can be redefined
  - do not take them for granted as working this way!
  - BASH can be reconfigured to change core behavior
- Format: cmd arg1 arg2 “arg3 with spaces” arg4.....
- First argument is always something to run, typically a binary program (doesn't have to be).
- All others are arguments to the program
- Programs can be separated by
  - ; & | > < space new-line
- All programs have a return code – 0 means “all ok”. Everything else is an error code

Example - Simple Command

- $ ls -l
  List files with details
- Program is “ls” - parameter is “-l”
- Where is “ls” and how did bash find it?
  - $ which ls

    /usr/bin/ls
  - BASH uses variable $PATH to search directories for an executable
    - $ echo $PATH

      /usr/share/Modules/bin:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin
    - Paths are separated by :
  - We can run a program using relative/full path – ignores $PATH
    - $ */usr/bin/ls -l*

```
[plarsen@localhost Pictures]$ ll
total 144
-rw-rw-r--. 1 plarsen plarsen 129454 May 28 21:34 file1.png
-rw-rw-r--. 1 plarsen plarsen  14067 May 28 20:49 file2.png
```

Running a Binary not in $Path

- Note the current directory is missing in $PATH
- Programs must be in PATH – or
  You must specify the path yourself
- Example – to exec in current directory:
  - $ ./hello
- Modify path in $HOME/.bash_profile
  - After modifying – run
  - $ source $HOME/.bash_profile

```
[plarsen@localhost hello]$ cat hello.c
#include <stdio.h>

int main() {
   printf("Hello World\n");
   return 0;
}
[plarsen@localhost hello]$ gcc hello.c -o hello
[plarsen@localhost hello]$ hello
bash: hello: command not found...
```

```
[plarsen@localhost hello]$ ./hello
Hello World
```

GLOBS AKA WILD CARDS

- BASH is responsible for processing GLOBs
  - ? matches any one character (or none)
  - * matches any characters including nothing
- Did you know there is advanced patterns?
  - ?(pattern) matches zero or one of pattern
  - *(pattern) matches zero or more of pattern
  - +(pattern) matches one or more of pattern
  - @(patterns) matches one of the given patterns
  - !(pattern) matches anything but the listed patterns

Very Special Matching Rules

- [ ] is a special way to create patterns using simple regular expressions
- **$ ls -d [e-h][[:alnum:]][[:digit:]]*.txt**
  Matches files that starts with e,f,g,h followed by any alpha-nummeric character, followed by a number and ending in .txt

Expansion done by BASH

- Before BASH passes control to the program identified, parameter expansion is done.
- When using parameters that include GLOB characters, make sure to escape them or they will not be passed
- Escape: \?
  The character ? - not expansion
- Using quotes changes expansion
  - “ “ - no expansion (globs) done. Variable expansion happens
  - ' ' - no expansion, no variables expanded

LOTS OF PARAMETERS!

- Commands can process a lot of parameters
- $ cat *.txt
  Outputs the content of all files ending in .txt in the current directory.
- The shell replaces *.txt with a list of fil
  calling cat

```
[plarsen@localhost demo]$ cat file1?
Hello10
Hello11
Hello12
Hello13
Hello14
Hello15
Hello16
Hello17
Hello18
Hello19
```

SCRIPTING

- Let the computer do the work for you
- Repetitive tasks should be made into a script/program that makes a single step out of many
- BASH is easily scripted. The commands typed on the command line can be inserted into a file and executed. No special cryptic stuff needed
- BASH offers a lot of control structures and features to manipulate data, testing conditionals, looping to make very complex tasks possible

A SIMPLE SCRIPT

- Create a file “compile.sh” with the following content:

```
#!/bin/bash
gcc hello.c -o hello
./hello
```

- Make the file executable
  - chmod +x compile.sh
- ./compile.sh
  Runs the script that compiles and then runs “hello.c”
- #!/bin/bash tells bash what kind of program is used to interpret the content of this file

Making Scripts More Generic

- What if we have a lot of programs and want to use the same script to compile and execute them?

```
#!/bin/bash
PROG="$1"
gcc "${PROG}.c" -o "${PROG}"
./"${PROG}"
```

- $1 = First parameter - “ “ are required if there are spaces in the file name
- PROG= defines a variable “PROG”
- ${PROG} returns the content of variable PROG

Important Shell/Bash Features

- All programs read from standard-in (stdin) and write to standard-out (stdout). All error messages are written to standard-error (stderr)
  - stdin = 0
  - stdout = 1
  - stderr = 2
- Streams can be created by coupling one programs stdout to anothers stdin
  - prog1 | prog2

Important Shell/Bash Features

- Output can be redirected to a file instead of the screen

  prog > file

- Input can be redirected from a file instead of keyboard

  prog < file

- You can suppress normal text but still get error messages

  prog >/dev/null

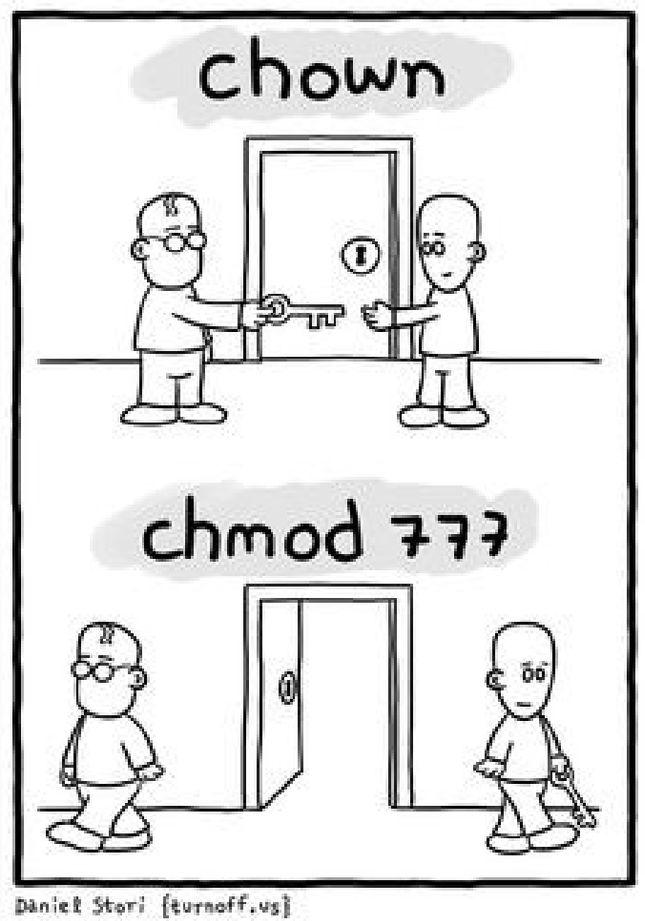
- To append instead of overwrite use >>

- prog1 | prog2
  Send the output of prog1 into prog2 as input

A few words about security

- Linux is a multi-user system. It's meant to run many different programs from many different users at the same time
- File security
  - Files are owned by a user (the owner)
  - Users belong to groups. A file is associated with one group (group owner)
  - Access is defined by owner, group and “others”
    - r – read
    - w – write
    - x – execute
    - These make up an octet r=4,w=2,x=1 – a total of 7 (octet)
  - chmod changes access settings for a file
  - chown changes the owner of a file
- All directories are files in Linux. Execute for a directory means “access but no peeking”.

Security made simple

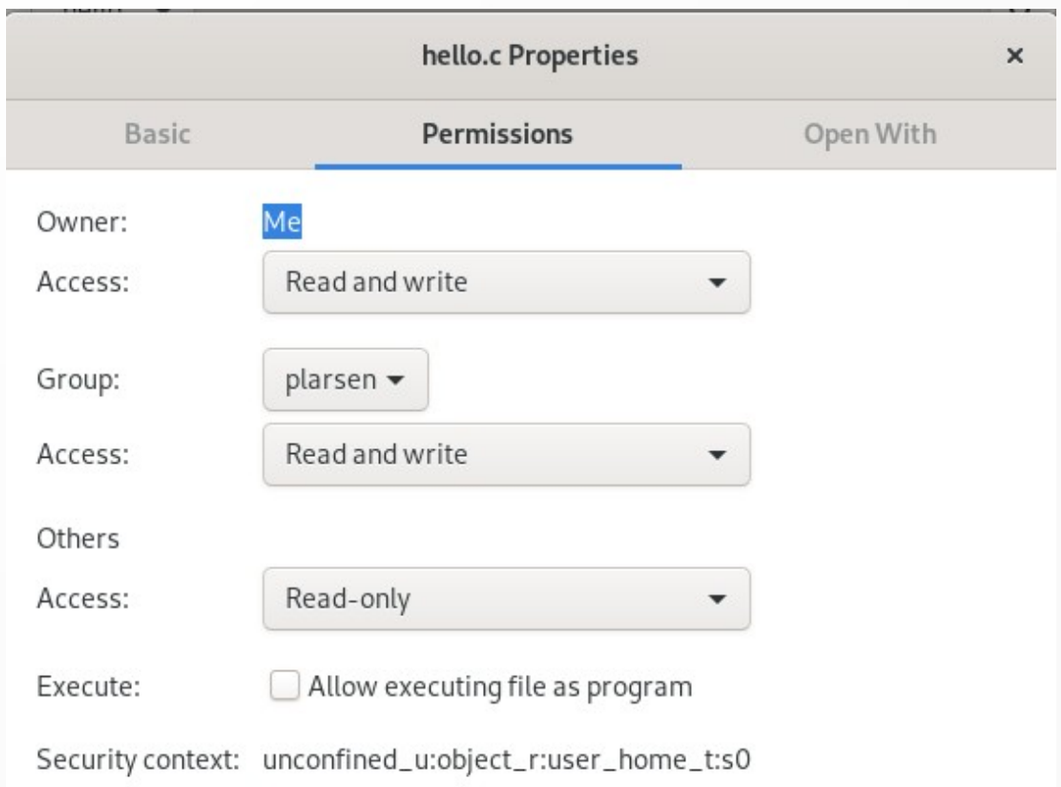


Additional File Security

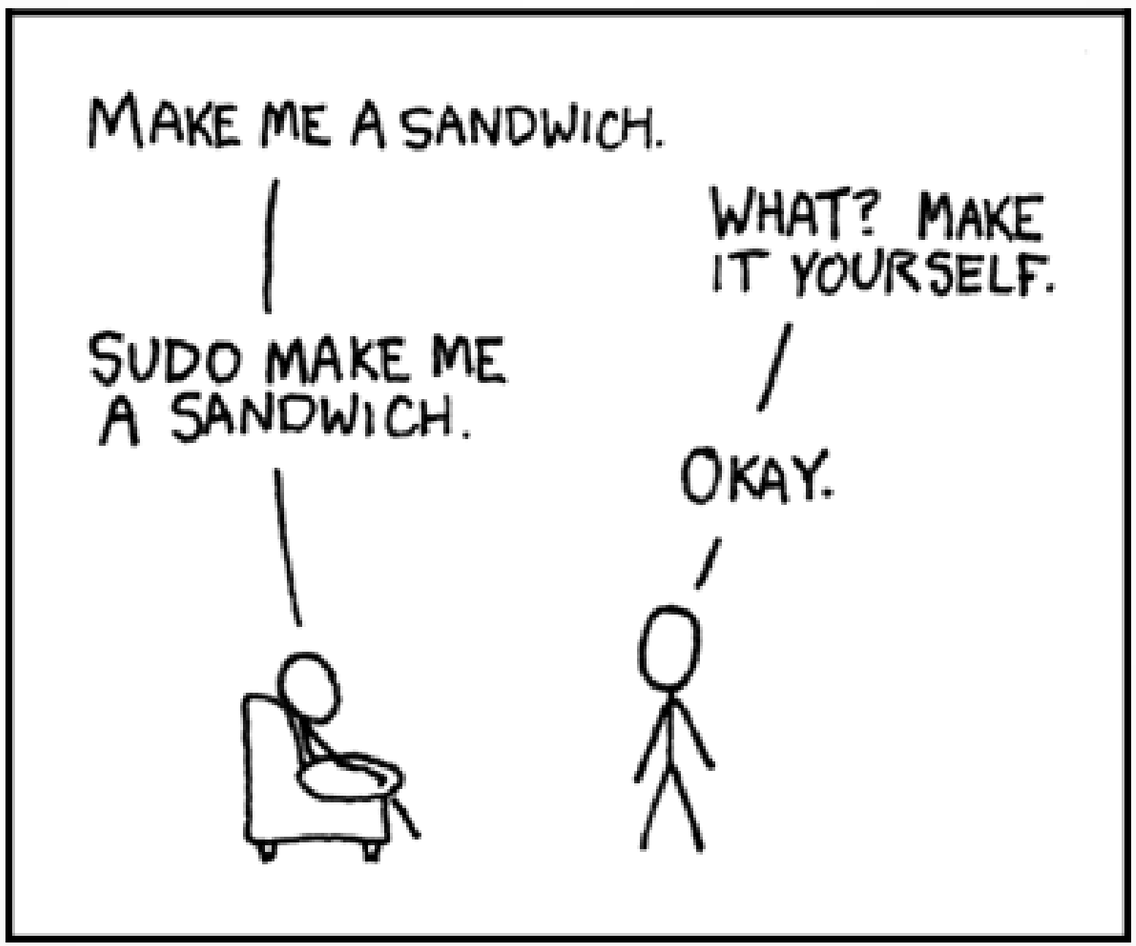
- FACL – File Access Control List
  - Rarely used – provides very detailed access lists
  - Must be enabled on the mount point/file system to be active
  - Can specify users by name, multiple groups, priority lists etc.
- SELinux
  - Is default on all distributions. Do not disable it!
  - Labels files with a security label – a policy engine controls which programs can do what operations with files of a given label.
  - This topic is not covered in detailed here.
    See https://people.redhat.com/duffy/selinux/selinux-coloring-book_A4-Stapled.pdf for easy to understand guide

File Security

- View
  - “ls -l”
  - “stat”
  - Nautilus (filer) in the GUI
- Change
  - chmod
  - chown
  - chattr
- A 'secret' 4th octet exists – beyond the scope of this talk



SUDO



Src: https://xkcd.com/149/

SUDO

- To execute a command with escalated privileges
- $ sudo systemctl restart sshd
- /etc/sudoers
  - Configuration file determining who can sudo, what a user can do and how
- A lot of systems defaults to allowing all users of the “wheel” group full sudo rights
  - # usermod -G wheel -a <username>
  - $ id # to check membership

Common and Useful Commands

- Our easy to install SMALL system has more than 1600 commands installed – this is not comprehensive
- basename, chgrp, chown, cp, date, dd, df, du, echo, ln, mkdir, cd, mv, ls, rm, rmdir, sort, stat, sum, tail, cat, touch, tr, wc, who, whoami, chroot, split, tee
- free, pkill, ps, top, vmstat, uptime, vmstat, which
- dmesg, kill, findmnt, lsblk, lscpu, mount, umount, sudo, blkid, fdisk, swapon, mkswap, swapoff, hwclock, mkfs

Long Distance Relationships

SSH and Pals

Access to a Linux Host

- All Linux systems provide terminal access
  - May add serial port terminal access like in the olden days
  - Virtual terminals created
    - Ctrl/Alt F1→F8
- Remote Access
  - SSH – Secure Shell
  - Telnet – nope, don't use it. Forget I wrote about it here!
  - VNC/Spice – Graphical
  - Through exposed services like a Web Server

Basic Usage - SSH

- By default, ssh server installed
  - May be disabled for security reasons
  - # systemctl enable --now sshd
- To access
  - $ ssh <servername>
- By default, uses the same username as current user
- SSH listens on port 22

SO WHAT'S DEAL?

```
[peter@cloud ~]$ whoami
peter
[peter@cloud ~]$ ssh dilbert
peter@dilbert's password:
Last login: Wed Jun  5 13:19:12 2019 from worklaptop.peterlarsen.org
[peter@dilbert ~]$
```

```
[ocp@ocpbastion ~]$ ssh ocp311
Last login: Thu Jun  6 23:09:03 2019 from 192.168.11.70
RedHat 7.6 x86_64

FQDN:         ocp311.ose3.peterlarsen.org (192.168.11.65)
Processor:    Intel Xeon E312xx (Sandy Bridge)
Kernel:       Linux
Memory Free:  1.26 GiB
[ocp@ocp311 ~]$
```

- Worst case, you enter password
- BEST case, a key authenticates you – no password
- SSH can run commands remotely
- SSH can tunnel network traffic
- And much much more

SSH TYPICAL USE CASES

- System administration
  - Automated/unprompted access to remote systems
  - Secure communication
  - Automation – Ansible would not work without ssh!
- Script access to remote systems
- Network access to systems behind firewalls

Setting up SSH keys

- $ ssh-keygen
  - Follow the prompts
- Copy the public key to the remote system
  - $ ssh-copy-id <remote host>
  - Login the old fashioned way
- Done – test it out
  - $ ssh <remote host>
  - NO PASSWORD!!

Access Behind Firewall

- I'm remote – need access to "secret-host" behind firewall. We have a bastion host "bastion" I can log into remotely
- $ ssh -L 8080:secret-host:80 bastion
- I can now access the secret-host:
  - http://localhost:8080/

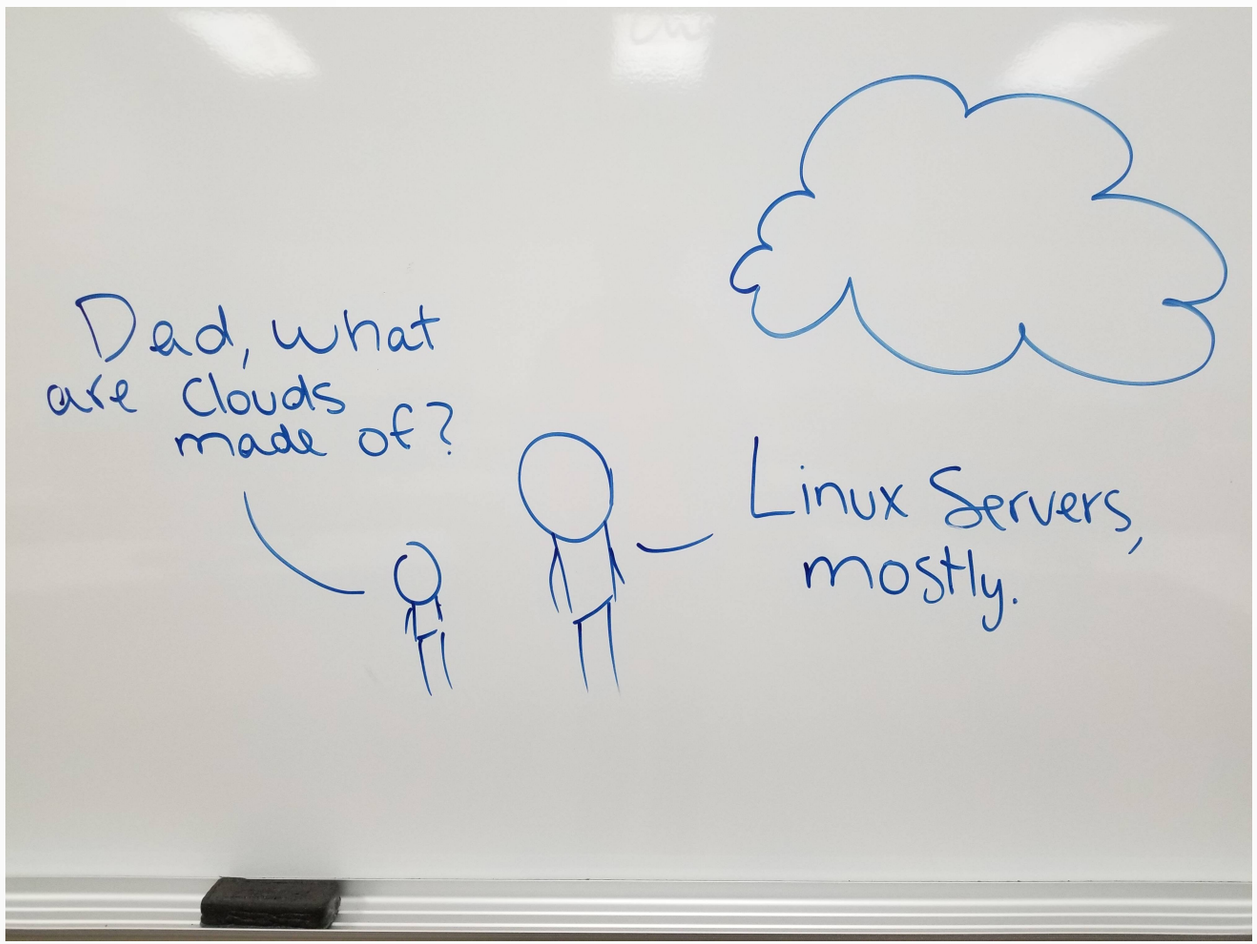
Other Examples

- $ ssh -l <user> <server>
  - Login as a specific user
- $ ssh -i <keyfile> <server>
  - Login using a specific keyfile
- $ ssh -vvv <server>
  - Get really verbose output on what's going on
  - Great for debugging

A FINAL WORD ON SSH

- In large enterprise environments, SSH will be locked down
  - No access to root
  - No password authentication possible
  - Strong keys required
  - Only specific hosts are allowed access for a specific user
  - Keys are centralized – not stored per host

Linux is everywhere!



Src: https://programming.codes/computer-programming-humor/

THANK YOU!!!

Reach me

ego.alter@gmail.com
@egoalter