true

Jeremy Hajek

Contents

1 Introduction to The Philosophy and Technology of Linux Part I

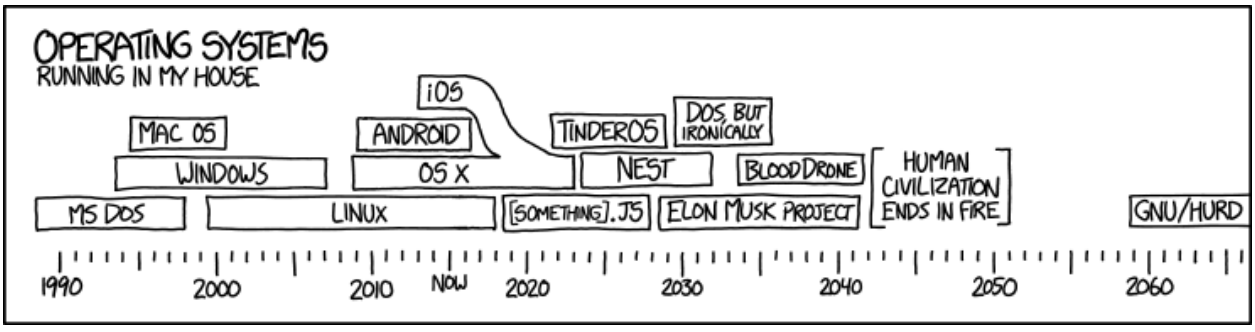

Figure 1: *One of the survivors, poking around in the ruins with the point of a spear, uncovers a singed photo of Richard Stallman. They stare in silence. "This," one of them finally says, "This is a man who BELIEVED in something."*

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. . . - Linus Torvalds 1991

1.1 Objectives of this book

As you know there are many many books on Linux out there. This book strives to be different in trying to highlight two areas. First the philosophy of Linux and its design. Anyone can teach you how to push a button, but this book aims to go beyond that and help you understand the reason the button is there in the first place. Above all Linux is merely a tool to help you get your work done. Seeing as this book is self-published its exercises and sections can be updated rapidly to cover the ever-changing tools in Linux. I have found over the years understanding the history of Linux goes along way to understanding how it works. This book strives to be different as well. In addition to being pegged to the LPIC-I/Linux+ content it will go beyond being just a training book.

For instructors - we are going beyond the traditional powerpoint world, and looking into using new technologies like Microsoft Sway for presentations. Also including audio and video excerpts as well as working code examples using doitlive for demonstration purposes. The book is being publsihed on Github under a Create Commons CC-SA Share alike license – this way as things change code can be updated or removed and new digital versions can be published quickly.

Some of these chapters are heavier in content then others. Some are lighter. Feel free to combine and remix the chapters. You are even welcome to fork the proejct on Github and remix it or contribute back patches and pull requests.

1.2 Presenatation Style

This book doubles as a universtiy text book. It has a 16 week semester structure, along with review questions, podcast assignements, and lab exerciese. But the book can be used beyond that. There is a wealth of other material that is can be covered in depth depending on your needs.

1.3 Cover Image

Will get to that once the content is done...

1.4 Background of Author

My name is Enigo Montoya...

1.5 Thanks

Professor Sam - who taught me that it is pronoucned *"etc ef-stab"* not *"etc ef es tab"* and how to read error messages. Professor Ray Trygstad who gave me my first real IT job and showed me the wonders of Perl. Illinois Institute of Technology who has entrusted me with much. My wife and kids who supported me always.

2 History of Unix and Linux

If you still don't like it, that's OK: that's why I'm boss. I simply know better than you do. Torvalds, Linus(1996-07-22)

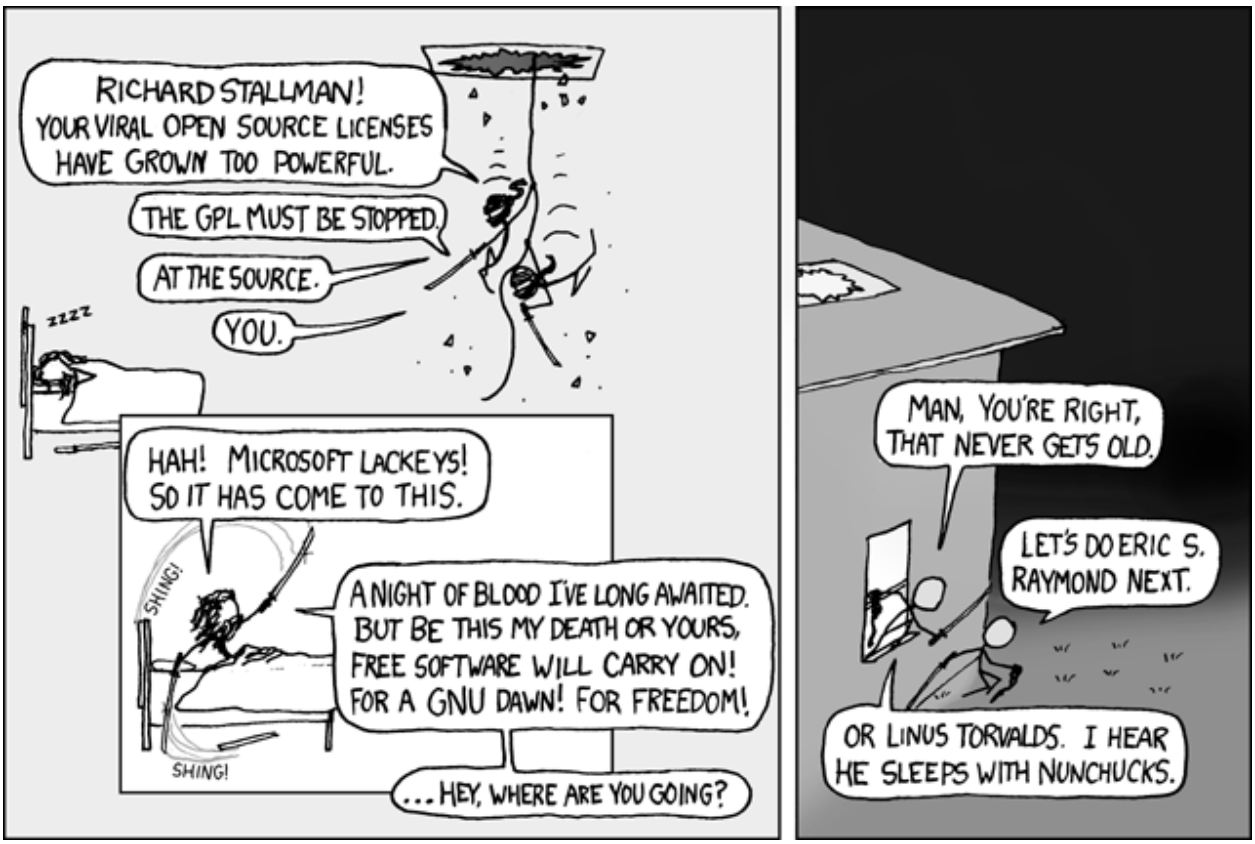

Figure 2: *At the end of class you will find this cartoon funny.*

2.1 - The Foundation of Unix and Linux

Why are you learning about Linux? It is a term that seems to be on everyone's lips. There is a good chance that you even have Linux running in your pocket and don't even know it! Raise your hand if you have an Android based phone or tablet? Here is a hint, Android Operating System is based off of Linux. So this chapter begins the start of your mastery of Linux. In addtition to teaching you technology, this book aims to teach you about the history and philosophy of Linux so you can understand where it came from and where it is going and why you are using it. Some pieces of this book will seem frustrating, after all the roots of the design decisions of what we are using today are 30-40 years old in some cases. Helping to understand what kind of technology was available and what these creators were thinking will help you master the concepts of Linux.

Chapter 2 Objectives

- Understand how the Unix Operating System was created
- Understand the contributions of Ken Thompson and Dennis Ritchie to Unix
- Understand the contributions of Richard Stallman to Unix, Linux, GNU, and FOSS
- Understand the contributions of Linus Torvalds to the creation of Linux
- Understand the nature of modern commercial implementations of Linux
- Understand the principles of the Linux Community and what the *"free"* in *"Software Freemom"* means

Outcomes

At the completion of this chapter a student will understand and be able to discuss the environment in which Unix and Linux were created. They will be able to relate key names; *Thompson, Ritchie, Stallman, and Torvalds* to their technilogical contributions. They will be able to understand what Linux and Unix are and how they relate to FOSS and commercial softare.

Convention Note

You will notice that I have been using the terms Unix and Linux interchangably so far. For a large part of this book the conventions are the same - their history is intertwined. Though this book focuses on Linux we would be depriving you of the full truth if we left Unix out. For this first chapter then we need to understand their related history. There are differences and there are similarities.

2.2 - Where it Began and Why it Matters Now

When we say *"Unix"* we are referring to an entire operating system. An Operating System can be boiled down into three main parts.

1. **Kernel**

Unix includes a kernel - a hardware abstraction layer that handles all the interfaces from the operating system to the hardware. The kernel is the portion of the Operating system that allows you to write once and interface with hardware through drivers. Otherwise you would have to compile the operating system everytime and include the hardware drivers for your hardware. Take Windows for instance - you have just one version for all of the hardware out there. Intel Chips, AMD processors, Marvell, Broadcom, and many others. But there is no need to buy a version of Windows, Mac, or Linux specific to that hardware - software drivers are inserted into the kernel and allow Windows, Mac, or Linux to simply communicate with the hardware. Think of the Kernel liek the engine of a car.

2. **User Interface and Operating System Tools**

All operating systems need a way for a user to interface with the kernel. This is where the Shell and User Applications come into play. The Shell is a way for you as the user to send commands to the operating system–which executes these commands through the kernel. Unix originally didn't have a graphical user interface, its time of being developed in the 70's precluded this. Once CRT monitors became prevelant the commandline shell became the standard interface. This allowed you to type commands direclty on a screen and see the results back–no paper involved. Eventually the X Windows System came along giving you the familiar desktop windows you are most likely used to, allowing for mouse and keyboard input. X Windows is the standard windowing toolkit that prety much all versions of Linux GUI build upon today.

Operating System tools include simple tools you take for granted like copy, delete, move, make directory, kill a process, open a text editor to modify a file, issue a compile command to the C compiler, or redirect output from the screen to a file. All these tools and more are included in Unix as an operating system.

User applications like a web browsers and email clients are seen as tools that are created by the user that just repurpose the existing Operating System tools and are built/installed by the user.

3. **Programming Language and Compiler Tools**

In the modern GUI comuting world we are used to just clicking on .exe or .dmg files and off our installation of Chrome or Firefox goes. In the Unix world all software is built using the C language. You needed a compiler to build the kernel, operating system, system applications, user tools, and any additional tools you create. Without a C compiler you cannot build or make anything. Today most code is pre-compiled for you and you can use other languages, but in the early days of Unix and Linux a C Compiler was neccesary as you were building the kernel, operating system, and tooling from scratch.

Linux is the same as Unix but...

Linux on the otherhand is technically not a full operating system like Unix. It is actually just a kernel, and is missing all the other tools listed above, though you will hear people refer to it as an operating sysetm. Depending on your audience you need to know both facets. The Linux kernel plus someone elses User Interface and Operating System Tools and Programming Language and Compiler Tools and premade user applications makes up a Linux distribution or simply Linux distro. Every company and person can equally contribute to the Linux kernel and make their own distributions. Linux was built out of the Unix world, keeping all the same conventions and ideas from Unix but starting in a different place. I think a good analogy would be the American Colonies in 1776 - they thought of themselves as Europeans, they came out of Europe but yet were starting new in America.

Take away point

Who uses Linux today?

- Facebook
- RedHat
- Oracle
- Google
- Amazon
- RedHat
- NYSE- New York Stock Exchange
- CME - Chicago Mercantile Exhcange
- IBM
- Android
- Microsoft
- Pretty much every top website except Stackoverflow.com

The question is not who uses Linux but the question should be when did you last use it? How did it get this way? Where did it come from? Ignore this part at your own peril, you will never understand Linux unless you understand UNIX at its core philosophies. Here we go.

2.2.1 - Thompson, Ritchie, and Bell Labs

Many people supported and worked on what would become known as Unix but two names have received most of the credit for the creation, promotion, and use of Unix. ***Know these names.***

Ken Thompson and Dennis Ritchie

Without Thompson and Ritchie, there would be no Unix and most likely no Linux today. Until recently both were hired as Distinguished Engineers at Google. Dennis Ritchie passed away in 2011. Ken Thompson is still working and recently help produce the [Go programming language](https://en.wikipedia.org/wiki/Go_(programming_language) from Google.

To begin We need to go back to 1968. The Unix project got it start in something called MULTICS, which was an attempt to build a multi-user operating system. At the time, this combined all the brightest minds of General Electric, MIT, ARPA, and Bell Labs. Now today those aren't names you think of when you think of computers. Yet in 1968/1969 General Electric and the government (ARPA) were the large funders and suppliers of computing (The PC market we know of today doesn't come until 1984!).

Bell Labs was basically the *"idea shop"* of the entire country - where the best and brightest went to invent everything we take for granted today. Bell Labs was spun off by AT&T and became Lucent Technologies, which became Alcatel-Lucent and now is soon to be part of Nokia. One could argue that Google and Facebook have taken its place where the brightest minds go to invent new things in America. No wonder that Dennis Ritchie, Ken Thompson, Brian Kernighan and even James Gosling (creator of the Java programming language) are and were employees at Google.

Like all projects that try to do to much, MULTICS stalled in gridlock between the different companies and the demands of the government. This left one crafty engineers with much free time and (for those days) a true rarity - unused copmuters; PDP-7s to be exact. Ken Thompson had an insiders view of the innovative things MULTICS was trying to accomplish and why the inner workings of the MULTICS project went wrong. Thompson also had a job to do as a Bell Labs researcher. On his own time and down time, he began to use these PDP7's and program his own multi-user operating sysetm, but with a different twist. It was designed by him, and solved daily work and coding problems he had. This operating system and its tools was then a project to help him get his own work done more efficiently.

PDP7

For this time, 1969-70, this is something radically new. Thompson had no idea that his pet work project was going to become part of a computing revolution. Where as MULTICS and other computer systems were designed by committees and based on marketable features–due to the nature of the up front financial investment, Unix was simple and easy to build because it solved only a small set of problems–which turned out to be the same problems every engineer had. The overall reason that Unix took hold was that it was designed by engineers to solve other problems that engineers were having–enabling them to get work done. This was pure genius and this is how Ken Thompson's mind worked.

Unix differences from existing commercial Operating Systems

- Written by Ken Thompson on his spare time
- No company owned it or committee designed it for commercial purposes
- Solved problems that engineers had
- Built by engineers
- Had a consistent design philosophy
- Designed to be portable and work on many hardware vendor platforms

Thompson's Unix success was also a by product of its main design philosophies:

- Everything is a file
  - This means that everything can be read from or written to: All the way from devices to text files

- Unix Portable – everything written in C
- Unix is a collection of small tools that do one and only one thing well
  - To build complex tools you chain input and output of tools together with *pipes* -> "|"
- The only file format used in Unix is plain ASCII text
  - Yes, there are compiled binaries but you generally are not reading and writing directly to them

Between 1970 and 1974 Unix grew in its maturity. And one of its crowning achienvements–its portability came to life. Unix was originally written in assembly lanugage for the PDP-7. It needed to be as low level code as possible because disk storage space was a HUGE premium in those days. This was good, but the problem with writting in low level assembly means that the code is optimized to only run on a PDP-7 system. Not on a PDP-11 or a DEC Vac, or an IBM 360, etc, etc. So what you gain in efficiency you lose in portablitiy. What good is Unix if it could only be used on a PDP 7? It would have stayed a Bell Labs pet project.

While Thompson was builing Unix his fellow engineers at Bell Labs got wind of what he was doing and asked to have access to his system, and then to be able to contribute additonal functionality. Enter Dennis Ritchie, who championed Ken Thompson's Unix in Bell Labs. Ritchie was a computer language creator and saw the utility of Thompson's Unix, but realized it was trapped in PDP7 assembler language. Today we take for granted high level lanugages like C, C++, Python, and Java. In the early 1970's these did not exist. Ritchie's initial work was on a high level language that could be built in order to compile and run the same code on two different operating systems. His initial work was on a [language called "B"](https://en.wikipedia.org/wiki/B_(programming_language "B Lanugage") which was derived from a language called BCPL. B was designed to execute applications and operating system specific tasks but didn't handle numeric data (a feature actually to save precious harddrive space). B was also missing many other features you would expect in modern programming languages.

What happened was that Thompson and Ritchie went to work extending "B" with all the features they would need to make an operating system fully function and portable. They called this language surprsingly, "C" – the same "C" lanugage you know today. C was different from assembler in that is resembled assembler code syntx had a high enough level of abstraction that the "C" code was an independant language. With the advent of C - Unix was rewritten in this language. With the creation of C compilers for different hardware, Unix could now be built and be recomplied on different architectures, PDP-7, PDP-11, DEC VAX, DEC Alpha, IBM 360, SUN SPARC etc, etc.

```
controller@controller-VirtualBox-node1:~$ cat hello.c
#include<stdio.h>

main ()
{
  printf("hello world");

}
```

```
controller@controller-VirtualB
        .file   "hello.c"
        .intel_syntax noprefix
        .section        .rodat
.LC0:
        .string "hello world"
        .text
        .globl  main
        .type   main, @functio
main:
.LFB0:
        .cfi_startproc
        push    rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        mov     rbp, rsp
        .cfi_def_cfa_register
        mov     edi, OFFSET FL
        mov     eax, 0
        call    printf
        pop     rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu
        .section        .note.
```

Since Ritchie created "C" to solve all the problems Unix had – it became the defacto lanuage of Unix and from that point on pretty much all operating systems are designed in "C" after Thompson and Rithie showed that you could use a high level lanuage to make Unix portable accross many platforms.

Brian Kernighan

Thompson didn't have a name for his project initally, another realted figure, Brian Kernighan, can be credited with giving it the name UNIX. This was a play on words– MULTI vs UNI in the name. Kernighan also helped write the original C language text book along with Dennis Ritche (called K&R C – some of you might have used when in school).

Unix Maturity

By 1974/75 Unix is growing in its maturity. Other Bell Labs divisions get wind of this and begin to request "tapes" for their own use. Tapes meant giant mounted magnetic tape reels that contained all the operating system installation code. By law AT&T was prohibited from getting into the computer business so they could not turn this into a business. AT&T left it curriously as Thompson and Ritchie's pet project. Many Universities at this time–wanting to teach computing and operating systems began to request copies of Unix to teach in their Operating Systems classes. This was attractive to universities because Unix was a fully operational and working system but the main draw was that the source code was freely given away by Ken Thompson. You sent him a letter, paid for shipping, and you got a reel within a week or so. Thompson had no concept of "ownership" and freely shared his project with anyone who was interested.

In 1975 Ken Thompson took a sabbatical from Bell Labs and went back to his Alma Mater, Berkly, in California. Installing the Version 6 Unix Release. The students at Berkley loved Unix and started adding their own features to improve it to solve their own problems. One student in 1978, Bill Joy, added vi and the

C Shell (two things still in use today in modern Linux) and started redistributing his "re-mix" of Unix called BSD (Berkely System Distrubition.)

By 1980, with many copies of Thompson's Unix now in circulation and nearly a decade of work you start to see fragmentation of the original Unix and many universities adding on their own customizations. Since the code was technically propriaary under AT&T's ownership - there was no way to contribute code back to the source. Unix starts to splinter. Another problem AT&T had was that by the end of the 70's all those students who had learned Unix in college went to work in corporations and began to request Unix be used on their hardware platforms at work. Unix was the only operating system of its type kind that could do this. Now AT&T had a financial motive to commercialize Unix. By 1982 AT&T released Unix System III, followed by System V in 1983, as a commerical product outside of Bell Labs for sale to commerical companies, while adding a multi-hundred dollar academic fee too. At this time the Berkley System Distribution of Unix was beginning to vary widly in functionality from commcerial AT&T UNIX. You see derivates of Unix like SunOS and SCO Unix being released as commerical companies based of BSD Unix. With HP/UX and IBM AIX being based on AT&T System V. The focus of Unix takes a dramatic shift now as the implementation portion is finished. Now the spoltlight moves to users and application creation. Enter Richard Mathhew Stallman, known also as RMS a researcher at the AI Labs at MIT who moves the discussion of software and *"software freedom"* into the spotlight.

2.2.2 - Richard Stallman and GNU

Before beginning anything in dealing with Richard Stallman there is one critical concept you need to understand: "software freedom." Often times you hear as a selling point that Linux is a good operating system to use because it is free – as in there is no cost. Stallman is not advocating about cost or the ability to charge money for software. This is a common confusion. He is advocating that the software itself is *"free"* as in freedom and that the user has the freedom to modify and or inspect its content and redistribute the original source. Ricahrd Stallman belives this is not a question of legality but of moral consequence and thereby will not use any non-free software period. When dealing with Stallman this is the fundamental fact you need to know.

Ricahrd Stallman was a student and a researcher at MIT in the early 1980's. He was part of what you would call today a hacker culture that was constantly researching and developling new tools and applications. As Richard Stallman progressed in his time at MIT he began to encounter events that he saw as counter intuative hacker culture that had been created at MIT by 1984. The spirit of Ken Thompson and the free and sharing culture of Unix was strong with these Jedi. Small things such as the addition of usernames and passwords on the school computer networks were seen as obstacles. Stallman saw the removal of the capability to modify a network printer's firmware to send an email feature in case of a paper jam as well as the beginning of propriatary software lock in–hindering the ability of the users to inspect the code and improve it to serve their needs. By 1984 AT&T began to withhold the source code of Unix and restric access of those in the academic world to be able to use the AT&T source code. By 1983 Stallman began to argue that the users of the software's freedom were being trampled on. Users were now beholden to the closed nature of the prodcuts they were using–even if they had paid for the software. Stallman saw this as more than an inconvienience and set about making it his life's work to rectifiy this issue.

GNU Manifesto

A plan began to hatch in his mind. Since Unix was the popular operating system at the time Stallman would make a call, a manifesto, for his GNU project to basically reverse engineer all the funcitonality, capability, and tooling of Unix–**BUT** without the propriatary and restricitve licenses with the source code being freely shared.

He felt strongly enough to announce the GNU project publicly in th fall of 1983, and to quit his job working in the MIT lab in 1984 to avoid conflict of interests and persue this work. Richard wrote his GNU manifesto stating his plans in Oct 3rd, 1985 and issuing a general call to arms in the GNU Manifesto. Here is the opening paragraph from the manifesto.

Why I Must Write GNU

> *"I consider that the Golden Rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. For years I worked within the Artificial Intelligence Lab to resist such tendencies and other inhospitalities, but eventually they had gone too far: I could not remain in an institution where such things are done for me against my will."*
>
> *"So that I can continue to use computers without dishonor, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free. I have resigned from the AI Lab to deny MIT any legal excuse to prevent me from giving GNU away."*

GNU is a recursive acronym meaning *"GNU's not Unix."* Stallman wanted to let people know his project was Unix like in fucntionality but not goverened by Unix's restricitve licensing. His passion was to develop a fully free operating system so that everyone who could use a computer could have access to a *"free"* operating system. Stallman is a brilliant man who had the capability to build an OS from scratch, but the project became more than a one man job.

Free Software Foundation and GPL

In late 1985 the FSF – Free Software Foundation was formed to be the holder of all the intellectual property of the GNU project. By 1989 a new role for the FSF arose. Commercial entities were starting to take notice and corrupt the idea of free software and the FSF needed a way to protect their software freedoms. But how? Copyright was restrictive and would make the FSF no different from the commercial entities that restrict freedom. Public Domain offered no legally enforceable protections. So enter Copyleft–which is a hack on the term copyright in the form of the GPL - GNU Public License. It flips copyright over by basically saying, you need to freely distribute the source code of any project lisenced under GPL, you need to attribute where your source came from, and that you cannot restrict any of these rights to any future derivative work. Meaning that you cannot close source some open source code that is GPL'd. This is different from the *"permissive open source licenses"* BSD, MIT, and Apache licenses which can have derivative works closed source.

GNU HURD

All seemed to be going well. RMS started this work of creating a free Unix-like operating system. Since Unix was built upon the C programming Language, the first thing needed to build a kernel, a shell, and tools was a C compiler. This project was called GCC or GNU C Compiler which was an *"free"* version of the propriatary Unix AT&T "cc" program or C Compiler. Next Stallman needed a text or screen editor to edit files and run his compiler tools to build software. In 1981 James Gosling has created the Gosling Emacs editor. James Gosling would later go on to create the Java programming language while at Sun in 1994. Stallman almost single handedly rewrote all of Goslings code in gmacs to produce a *"free"* version of Emacs - now over 30 years old and still in use. The project continued on to now basic Operating System commands such as ls, grep, awk, make and ld. Eventually an entire "zoo" of projects were created and are listed and described here: other GNU tools were added by contributers and volunteers. They did remarkably well and had reverse engineered and in some cases improved the components of Unix by 1991 (8 years of work). They built everything except 1 critial piece... they didn't have a kernel for their operating system. Turns out that writing a kernel is much harder than it looks.

A project was started called GNU Hurd to be the kernel for the GNU operating system in 1985. The term GNU Hurd is also another clever recursive hack. Since GNU has a double meaning. There is a large goat like animal called a gnu, which lives in herds that roam the plains of Africa. The name HURD came from the idea of a herd of animals and the fact that the design of the GNU Kernel would be a "herd" of small micro-processes comminicating together. It seems that GNU hackers really loved cleaver hacks. It it something that you have to get used to in opensource as the spirit of bad puns and clever hacks has carried on to this day.

Hurd made some false starts in its initial micro-kernel development phase causing multiple versions to be created and scrapped. What they were trying to do was really innovative but really complicated to have it work reliably. In retrospect HURD was never finished. By 1998 The GNU project had all but stopped active development and promotion of GNU HURD as the kernel for its free operating system. The GNU project realized that the Linux Kernel had accomplished what GNU was trying to do in a matter of 4 years and in a more clever way.

GNU HURD is currently in a usable alpha stage and downloadable today by joining it with the Debian Linux distribution applications–all GPL approved mind you. Instead recommends the Linux kernel instead. In someways this was the realization of Stallman's dream and yet someways this was his biggest disappointment that Linus Torvalds and not the GNU project finished the kernel. By 1991 the Linux kernel pops onto the scene and we have another little revolution in the free and open computing world. Thompson -> Stallman -> Torvalds.

2.2.3 - Minix, Linus Torvalds, and Linux

Minix

Before we talk about the Linux kernel, we need to talk about the Minix operating system. With the closing off of the AT&T Unix source code by 1984 to academics and researchers in the university - they were left without source to show as examples. One professor Andrew S. Tanenbaum teaching at Vrije Universiteit in Amsterdam - began to write and implement his own Unix-like operating system but only for teaching and demonstroation purposes. It was 12,000 linees of C code and system call compatible with commerical Unix. The name Minix was a combination of "minimal" and "Unix." Minix 1.0 and 1.5 were released in 1987 and 1991 respectively with the original purpose as only a teaching tool. Minix 1.0 and 1.5 were freely avaialable to anyone as the source code came in the appendix to a text book about operating systems written by Tanenbaum in 1987. Minix was designed to run initall on older x86 Intel processors and in version 1.5 Sun Sparc processors. These were common desktop stations in use at the university at that time. Any enterprising student could find and old 8086 PC or old Sun Sparc Station to run it on. The source code for Minix 3 is currently available in a git repository and is still being developed and researched. In 1991 many people believed that Minix could have been a viable alternative to commercial Unix and become the still missing GNU Hurd kernel. But the Minix creator, Professor Tanenbaum, was not interested in moving into this space and the code was no where near as mature as the Unix code base, which by 1991 had been in existance for almost 20 years! Minix appears on the radar but was not the missing piece to the GNU puzzle.

Professor Andrew S. Tanenbaum

Linux

Linus Torvalds

The Linux kernel comes to us from a graduate student named Linus Torvalds who developed it while at the University of Helsinki in Finland in 1991. As a student Torvalds was using Unix on the universities Sun Sparc Stations. He was not pleased with SunOS but felt it was the best of the commercial Unixes. His real dream was to set out to run his own Unix like operating system on his own personal PC. He had recently purchased an Intel x386 processor based desktop PC. Linus tried Minix, but was put off by its minimalist approach and realized it had some good design concepts but was not a complete Unix replacement. In a fashion not unlike Ken Thompson, Tovalds set out in the early part of 1991 deciding to see if he could build his own kernel for his own operating system for his own use and purpose that was Unix-like but wasn't Minix.

This was very impressive feat for a single person. Torvalds himself acknowledged that if GNU Hurd had been ready or if at this time AT&T hadn't been suing BSD, he would have re-used their kernel work and not built his own. By August 25th of 1991 the initial release of the Linux kernel was posted online. The quote from the beginning of the chapter was the basis of the initial post to Minix Usenet Newsgroup-(Bulletin board like precursor to the actual Internet - like Google Groups – today you would use twitter). His initial work was not quite a full fledged system but really just a small kernel, a user shell (GNU Bash), a C Compiler (GCC) but it was like a crack in a damn - it would only get wider and bigger.

By September of 1991 Linux kernel version 0.1 has been posted to the University of Helsinki FTP servers for public download. By Feburary of 1992 Linux 0.12 kernel had been released. At that time Linus decided to give the project a real license for its use. Having seen Richard Stallman speak at the University of Helsinki a few years back, Linus was inspried and dedided to release the Linus Kernel under the GPL license. This was the smartest thing anyone could have done. Can you see the connection to the GNU project? The reason we Linux is so popular is because of this fledgling kernel that worked enough for people to use, hack on, and build upon now had a governance structure that could accept code contributions and be avaiulable for commerical work as well. Being GPL the Linux kernel was instantly compatible with all of the entire GNU project's tools base. You instantly had the kernel that GNU was missing and the Linux kernel had all the tools and applications ready to be used.

People began downloading and compiling his kernel, adding GNU tools, and making a fully capable UNix-like operating systems. Linus' brilliance comes not from ingenuety but comes from good engineering pricipals of knowing when not to go down dead-end developenent trails. Torvalds work was not perfect but was good enough that others could take it and start to use it and improve it. From 1992 to 2001 Linux grew rapidly in size and features and spawned commercial companies to sell and support Linux Distributions. Stallman's dream was being realized.

There should have been cause for great celebration with Linux and GNU coming together. The FSF saw this as a victory for GNU and began calling the system GNU/Linux hoping that the FSF and free software would get the recognition it deserved. But Linus Torvalds didn't see it that way. He has a unique personality–perhaps a bit arrogant. He just ignored the FSF's requests and people referred to what should have been GNU/Linux as just Linus, leaving the GNU part out even though all of their tooling is what made Linux possible. This is a spot of contention with the FSF.

Linux Kernel unique attributes recap:

- Developed to solve one person's problem of wanting his own Unix liek OS
- Released often
- Accpeted contributions back
- Released freely and protected by GPL license
- Used existing GNU tools - no need to reinvent the wheel

Personality

Linus Torvalds has a renowned personality. Some people thing it is a schtick or a comedy persona he puts on. But he is very uncaring in relating to others and can be really mean and agressivly mean spirited to those who he has disagreemtns. When approached about this Linus states that he only cares about the kernel and no one else matters to him. These links below provide some points and counter points about Linus.

- Torvald's right to offend
- Torvald doesn't care
- Linus response to previous article

AT&T and BSD Lawsuit

From August of 1991 to Feburary of 1992 there was a rush of interest in Linux development? But where did all these develoeprs come from? Remember the Berkley System Unix Distribution? In the late 80's and now 1990s its development had been flourishing. It began to support features that not even AT&T's Unix had. BSD was such alarge project through that not all of the original code that was given the Berkly under acadmeic license had been reqritten. AT&T found its code in BSD Unix and took them to court. In early 1992 there was a court order development injunction preventing work from being done on BSD Unix. This was just the time that Linux kernel development, covered by GPL now, so there was no licensing encumberment, devlopment flourished. By the time the lawsuit was finished in late 1993/1994 it was too late. The Linux rocket had left the launch pad and was never coming back.

2.2.4 - Free Software vs. Open Source Software

By the year 1998 a new idea in the *"Free Spftare"* movement was rising. The long expected GNU HURD kernel never arrived, being replaced by Linux. You also begant to see a corporate interset in opensource. With all the best intentions the term *"free"* in free software had an overriding air of being only about cost. Many commercial entities were simply not interested or afraid because they were concerned about loosing the chance to make money or retain rights over programs they created. Some developers seeing a chance to promote the quality and community to the larger commercial markert. These developers did not hold with the FSF's moral stance on free software as the ultimate argument in free software adoption. Instead they compared open development and open code as superior in quality and cost in cost reduction to closed source development. They beleived opensource was a business process decision and not a moral decision. Enter Eric S. Raymond.

Eric S. Raymond

Eric S. Raymond is another developer considered a peer along with Ricahrd Stallman. Eric was one of the first to embrace the Free Software idea and promote using free software. He was so convinced by the free softeare and the open development method that took place with Linux that he penned a seminal paper that was later reprinted called, **"The Cathedral and the Bazaar."**

His main point was that by usual business practices - Linux should have been a massive failure and poorly implemented experiment. But instead it was an unprecendented success. His article examined why this is the case. His conclusion was that the open source code and open design methodology treating your user as a valued resource was vital to opensource project success. Based on this Raymond and Bruce Perens founded the Open Source Initiative (OSI). Their goal was to promote free software but instead of focusing on the moral issus they focused on the desgin principals as producing superior software. A quote from Raymond puts his opinion bluntly;

> *As head of the Open Source Initiative, he argued that advocates should focus on the potential for better products. The "very seductive" moral and ethical rhetoric of Richard Stallman and the Free Software Foundation fails, he said, "not because his principles are wrong, but because that kind of language . . . simply does not persuade anybody".* Eric S. Raymond

The "Catherdral and the Bazaar" was also influential in helping the Netscape Corporation opensource their Netscape Web Browser code as the company went bankrupt under the name of the Mozilla project. This code gave rise to what would eventually become the Firefox web browser in 2002–thanks to Raymond's writings. The OSI was willing to make freedom compromises in order to make larger productivity gains with opensource software. The FSF will not compromise. Richard Stallman fired back in his article Open Source Misses the Point The terms do overlap Free Software and Open Source but ultiamtely have two divergent meanings. There has been some compromise in the naming [FLOSS, Free Libre and Open Source Software] (https://en.wikipedia.org/wiki/Free_software_movement "FLOSS"), Free and Libre Open Source Software – but the FSF rejects any licensing that allows a user to restrict the use of "freedom" Code. One arguemtn would be the existance of DRM software. The OSI group would not be opposed to push for open sourced DRM software. But the FSF would be opposed to the entire concept of DRM–which is a tool they believe for restricting a users freedom.

You can read Raymond's two seminal books on Unix and Open Source philosophy online online as they are free:

- The Art of Unix Usability
- The Cathedral and the Bazaar

Linux makes you rich

As the 1990's went along we began to see eastablished companies adopting and using Linux. as well as the rise of commercial Linux companies. Of all the companies that started at that time RedHat Linux is and

was the most successful. Most of all of the Linux distribtions started pre-2003 no longer exist or are not commercially viable. To illustrate this, today (August 10th 2015) RedHat Linux has a market cap of ~14 billion dollars.

2.2.5 - The Rise of Commercial Linux and Modern Linux Distros

As the nature of Linux grew and corporations become more invovled in kernel development, the value proposition of Linux begans to grow as well. The combination of the Linux kernel and the GNU tools, plus GUI tools became known a Linux distribution - which anyone could freely make. The shortname became known as a Linux *distro.* Another term to be aware of is a there are different flavors, derivatives, or spins of Linux Distributions.

After almost 20 years of Linux we can think of the distributions mainly hailing from two distinct families: Debian and RedHat. There are many other quality distributions of Linux that I don't want to leave out or paint in a bad light. For the purposes of this book I will focus on the two main distributions. You can find almost all known Linux distributions at http://distrowatch.com

- Slackware
- Gentoo Linux
- SUSE Linux
- Kali Linux - Hacking tool based Debian spin
- antiX Linux - lightwight Debian derivative focused on old machines.
- Arch Linux
- Tails Linux - Online security focused Linux distro - debian spin
- LXDE - lightweight system focusing on reinvigorating older laptops.
- and many more...

Debian Family

> *"I founded Debian in 1993. Debian was one of the first Linux distributions and also one of the most successful and influential open source projects ever launched. Debian pioneered a number of ideas commonplace today, including employing an open community that allowed (and encouraged!) anyone to contribute (much like how Wikipedia would later operate). And, with its integrated software repositories anyone could contribute to, Debian arguably had the industry's first (albeit primitive) "App Store". Today, more than 1,000 people are involved in Debian development, and there are millions of Debian users worldwide."* - http://ianmurdock.com

The Debian family contains 3 major sub-families:

Debian Linux

The Debian distribution (pronounced deb-ian) was founded in 1993 By Ian Murdock and is unique for being one of the only non-commercial company backed Linux. The current release is Debian 8.1 codenamed Jessie, June 2015. The Debian project and its history can be found at http://debian.org and history of Debian.

There are currently 128 major Debian based distros according to distrowatch.com.

These are the main points of Debian and the key I believe to their long term success and usage accross the Linux landscape:

- Initial release scheudle was yearly but as Debian project has grown now is two year release schedule
- Releases are named after charactrers from the Toy Story movie.

- It is the only major distribution not backed by a corporation.

Figure 3: Debian Logo

- All volunteer project and organization – project leader is elected on a rotating basis
- Dedicated to protecting software rights and freedoms of users
- First major distribution to come with a Software contract - of what the project will guarantee to the user.
- Debian supports free and open source software as superior to closed source but will allow for closed source software/drivers to be installed by the user.
- Supported st various times 11 different processor types giving it a wide install base.

Ubuntu

Ubuntu Linux is a unique distribution. It is entirely based on Debian. It is Debian repackaged with a focus on applications "just working." Around 2004 Mark Shuttleworth, the founder of Ubuntu, was unnerved that Windows had such a domination of the PC market. He had been a Debian developer, but felt that the partial lack of a corporate sponsor in some ways hindered Debian from catching the marketshare from Windows. He set out to make a Debian based distro called Ubuntu. This is a Zulu word for *"community"* as Shuttleworth wanted Linux to be people friendly and work really well out of the box–like Windows.

By 2004 RedHat, who had owned the desktop Linux market realized that there was little money to be made in that market so they abandoned it decideing to focus on the enterprise market. This left a void that Ubuntu rushed to fill and they did it well. By 2005, Mark Shuttleworth who had started the Thwate SSL security company which was bought out by Verisign, took his money and invested 10 million dollars in the Ubuntu Foundation to subsidize the creation and maintainance of Ubuntu Linux.

Mark Shuttleworth

What made Ubuntu so succesful was tha they forked the opensource work of rock-solid Debian but built on top of it adding in closed source code and user centered features where neccesary in order to make the best experience. They had business in mind and have indeed captured the desktop Linux market. But one problem is they haven't found a way to make much money off of their excellent product. Ubuntu is basically "living" off of the initial 10 million dollar investment of Mark Shuttleworth. Shuttleworth formed a commercial company called Canonical was formed to handle commercial support and hires the developers who work on Ubunutu.

Ubuntu pioneered the idea of rolling releases - releasing every 6 months compared to Microsoft doing 3 to 5 years. Each distribution is released in late April and late October so there are two distrubutions per year. Ubuntu also introduced the concept of an LTS, Long Term Support - this means that certain releases will have security pathces, fixes, and software backported to it for 5 years, allowing you to base an enterprise business off of this product and have the stability you need. These LTS releases happen every even year and in the April distribution. So Ubuntu 10.04, 12.04, 14.04, 16.04, and so forth. (the first number being the year.)

Linux Mint

Linux Mint started also in 2006 as a fork of the Ubuntu project but with a different desktop interface. Linux Mint focused foremost on the user and desktop experience out of the box adding multimedia codecs for playback of audio and video directly to the install (see Flash). Linux Mint is even more user experience focused than Ubuntu and is one of the most popular Ubuntu based distros.

Devuan Logo Here

Devuan Linux

Devuan Linux Project (Pronounced *Dev-one*) is a fork of the entire Debian project - not just a Debian based distro. This is a result of a "Debian Civil War" with half of the Debian developers leaving in the Debian project in the beginning of 2015 to begin this distrobution from scratch. It is a direct fork with fundamental changes to the core operating system. Other distros change application look and feel but to change the core operating system is a monumental task. The state of the OS is in late Alpha or early beta as of August 2015 with vm images availalbe for download. We will talk about this more in detail under the topic "LInux Civil War" later in this chapter.

Some of the other notable Debian/Ubuntu based distros are as follows:

- Lubuntu
- Xubuntu
- Kubuntu Uubntu remixed with the KDE desktop Environemnt
- SteamOS Steam onlien gaming companies official Linux distro
- Kylin Linux Ubuntu Distro designed for Mandarin Chinese as opposed to English.
- [Raspian])http://www.raspbian.org/ "Raspian") This is a Debian based distro that is standard recommeded for the Raspberry Pi.
- gNewSense <- GNU/Linux FSF recommended distro, entirely GPL compliant software.

RedHat Family

Figure 4: RHEL

RedHat Linux was formed shortly after teh Debian project launched in 1995 Marc Ewing and Bob Young. It was one of the first commercial Linux companies and one of the few to survive to the modern day independant of an existing company. RedHat source code is currently shared accross three main distributions: Fedora, RHEL (RedHat Enterprise Linux), and CentOS.

You can read more about RedHat from their website:

- About
- Red Hat History

Fedora Project

The Fedora Project was started in 2003 when the Red Hat Desktop Linux product was merged with the comapny/community based spin off Fedora Core Linux. The Fedora project's focus was rapid development and rapid release. They would release two distributions almost yearly, with package and update support only extending back to the previous version cutting off support to viable but from Red Hat's point of view outdated software. Remember their focus was rapid iteration of the project to quickly test new technologies.

For example Fedora 22 was released on XYZ date, Fedora 21 was released ABC, and Fedora 20 was released DDD but is not suppoprted anymore - even though it was only realed a short while ago! Why so fast and so

merciless on not supporting older versions? With this concept they would not have to without having to worry about legacy applications. This distribution was meant for desktop users and developers who don't mind updating rapidly. The reason for this iteration is that the Fedora Project is really jsut a testing ground for technology that will eventually go into Red Hat's enterprise project, referred to as RHEL.

Currently there are 25 Fedora based distros or Fedora calls them *"spins"* – this term is unique to Fedora.

Figure 5: RHEL

RHEL

RedHat began to see the opportunity to create a Linux distro targeting enterprises and make money using opensource at the time. A big market that was practically cornered by two companies were Java based applications and database servers - MySQL or Orcale. These markets had been the domain of Sun and its Unix based Solaris Operating System for years, as well as Microsoft runnning Oracle on Windows. RHEL could enter that market, running the same applications, and do it on cheaper Intel x86 based boxes. With Oracle announcing it would port its products to RHEL, this platform became to the go to choice as the alternative against Microsoft and helped put Sun and Solaris basically out of business. The acronym stands for RedHat Enterprise Linux.

The key to RHEL's success in the enterprise is its long term stasbility. Much like the version of Windows Servers it competes with - the applciation platform is expected to run for 5+ years. A enterprise grade server product cannot be changing every six months like the Fedora project. RedHat instead takes "snapshots" from Fedora and freezes them in time. As of today (August 13th 2015) the current version of RHEL is 7.1 which is a freeze of the technology point in Fedora 19, whihc was released July of 2013. This way the developers get to know the platform and software versions that will be maintained and supported long term. How succesful is this strategy? By 2012 they became the first Linux based company to make a billion dollars in a physical year. But this success brought about a serious opensource question, if you have a successful prodcut like RHEL, since you are using GPL based opensource code–you have to opensource your code–that means anyone else can redistribute your code freely, in theory eating your lunch.

Centos

By 2004 many people began to see the utility and success of RedHat Linux, and being opensource they began to fork the code and make their own distributions. CentOS is one of them. By 2010 they emerged as one of the two remaining RHEL derivatives. Their developers, like Debian, are entirely volunteer based and not backed by a company. Their motive was to take the solidness of RHEL and just update a few features and add more modern software packages sooner then the 5 year RHEL cycle. Initially RedHat didn't support CentOS–taking them to court numerous times, as CentOS had not removed all of RedHat's trademarked logos in all the code. Eventually all of RedHat's copyrighted material was removed and CentOS has a leagal copy of RHEL to redistribute and use. This made RedHat angry as they were loosing sales to enterprises using CentOS insteaf of RHEL. By 2014, RedHat and Centos came to terms to work together–with RedHat offering to sell support contracts to CentOS users. Is CentOS doing anything illegal? Not according to the GPL and the spirit of opensource, but it does bring up the financial issue again.

Oracle Linux

Oracle Linux

Did you think that Oracle would allow their logo to be displayed under an open license? Not to be out done. Oracle who saw that many of their customers were paying RedHat for operating systems licenses, buying support contracts, and then running their database on top of it wanted a piece of the action. Oracle now owns Java–which is the primary tool used to interface with all the Oracle and its suppoprting products. Oracle made a fork of RHEL's opensource code and placed their logos, Oracle specific tools, and made their own software tweaks in this fork and called it Oracle Linux.

Oracle Linux was born in 2007 and is a fully GPL compliant OS. Oracle claims that their *"Unbreakable Enterprise Kernel"* is fully compatible with RHEL, and that Oracle middleware and third-party RHEL-certified applications can install and run unchanged. One may ask, isn't this illegal? Is Oracle breaking the law? Not according to the GPL - they are fully entitled to do this and thus compete with Red Hat selling support contracts on Red Hat's created software–this is the nature of the GPL license.

Unix and the BSD Family

While Linux was exploding in the mid 1990's the AT&T lawsuit against BSD had been settled and work could resume of the BSD forks of Unix. Unfortunately the BSD code splintered into 4 main distros pulling the already thin developer group that hadn't shifted to Linux development, even thinner.

Figure 6: FreeBSD

- Released in November 1994
- Essentially the inheriter of the BSD code base and the largest BSD implementation.

- Leagally prohibitted from using the term *"Unix"* as outcome of AT&T lawsuit.

- Fork of FreeBSD in April of 2005 by Matthew Dillon.
- Focused on unique techniques in handling mutliprocessing in the FreeBSD kernel
- Introduced a new filesystem called HAMMER and HAMMER2

PC-BSD * FreeBSD based distro with a focus on user interface and experience. * Provides friendly installers and package managers for users

Ghost BSD * Does something...

- Released October of 1994 as another version of the BSD code after the lawsuit.
- Focuses on portability to run this OS on nearly every platform you can think of.

- Fork of NetBSD lead by Theo de Raadt end of 1995
- Founded by Theo de Raadt
- Theo was banned from NetBSD in 1994.
- He complained that they were developing too slow and not focusing on security.
- OpenSSH comes out of this project.
  - Microsoft recently became the first "gold sponsor" of the project
  - Recognizing the standard of SSH (secure shell) they are moving to port and integrate SSH natively to Windows.

Figure 7: DragonFly BSD

Figure 8: NetBSD

Figure 9: OpenBSD

- Project is focused on radical implementations of security and safe coding practices–leveraging itself as the most sercure OS.

Minix 3

- Released October of 2005
- Since then the OS went from a teaching tool to a product being used commercially.

- Began using NetBSD user space applications to give it a GUI and make it a viable commercial product.

Open Solaris / Open Indiana / Smart OS

Open Solaris

Joyent

SmartOS

- In 2006 Sun had experiemented with creating and opensource user based distro from their Unix based Solaris OS
- They hired Ian Murdock (the guy who started Debian) to oversee this project
- Project was called Open Solaris but was killed when Oracle purchased Sun in 200X
- The resulting codebased was forked and turned into a project called Lumios
- Currently their is a competing Open Solaris based distro called Smart OS which is produced by Joyent
  - Combines the best of the BSD underpinnings but runs the best of Linux based desktop applications and software

2.2.6 - Impending Linux Civil War

Lennart Poettering

Not since Linux Torvalds has a man been so loved or reviled in the Linux community. Lennart is a name you need to know as well. He is currently a developer for Red Hat and having worked on many open source

proejcts. His current project is systemd. Systemd is a replacement for the traditional SysVInit program that started all of the Operating Systems process upon boot.

Poeterring has angered many people by breaking certain Unix traditions and conventions in the name of speed and features. The Unix philosophy of having little programs do one thing well goes right out the window. Poetering argues that philosophy is a byproduct of an era where computing was slow and disk space was precious. If Linux wants to be taken serious like Mac and Windows–it needs to think like Mac and Windows. Poeterring is young and wants to push the development of the operating system of Linux forward rapidly.

The other major point of contention is with all the changes in systemd to the boot process, many other pieces of software need to change as well. Linux has always been about choice but the GNOME desktop developers have chosen to hard integrate with systemd. Meaning that if you operating system uses systemd instead of SysVInit - then you m,ay very well in the future be forced to use GNOME as it will become a dependency of your init system.

This leads to an interesting point. All major distros have moved to systemd. Debian was the last hold out and they actually had a civil war and split over this issue. Half of the developers left and went to form a distro called Devuan–which is focusing on removing all the systemd dependencies and putting choice back in the users hand.

Systemd has many nice and needed features. Leonart is updating pieces of Linux that haven't been touched in ages. He even wrote a 21 part defence of systemd on his website. I will talk more on the technical aspects of systemd in the chapter X.

The fears of Linux users are that systemd will grab dependencies and eventually force Linux users into a small sub-section of systemd supported software choices. In a sense create a vendor lock in. What makes this all the more intreguing is that Lennart works for Red Hat. Would Red Hat mind if this systemd technology improved the Linux experience at the cost of choice of software and freedom available to the user? That is a good question. This also begs the question - can Linux survive as an independant and open software or does it need a commercial comapny backing it? Or could this be seen as Red Hat's grab for the entire Linux market? It is too early to tell but keep a watch on what happens with systemd.

2.3 - Chapter Conclusion and Summary

Wow - we covered a lot of history – but it is important to the understand the current state of Linux usage. Learn more about opensource licensing

Additional Reading on the Unix history side

2.3.1 - Review Questions

Get into groups and answer/discuss these questions

1. Based on the movie's tone and time - why would you think there is a definate anti-microsoft tone?
2. What influence did Bill Gates' 1976 *"Open Letter to Hobbyists"* influence the opensource movement, if any?
3. Would Richard Stallman enter into a discussion on which is a better product: Microsoft Word or LibreOffice Writer? Why or why not?
4. Would Eric S. Raymond enter into a discussion on which is a better product: Microsoft Word or LibreOffice Writer? Why or why not?
5. Why did Bruce Perens help write the Open Source Definition / Debian Social Contract Standard?
6. What were the two commercial Linux companies featured in the movie?

7. What is RedHat Linux's stock price today?
8. What is VA Linux's stock price today?
9. How does Ricahrd Stallman react at the end of the movie of the success of the Linux kernel to the exclusion of the GNU tools?
10. As a follow up - why do you think this is so?

2.3.2 - Podcast Questions

Listen to the FLOSS podcast number 73 with Tim O'Reilly - http://twit.tv/floss/73

- Who is Tim Orielly? ~3:00-5:00
- What is Oscon? ~6:45
- Who coined the term web 2.0? ~13:34
- What did we learn from the IBM PC? ~18:30
- What is web 2.0? ~19:30
- Open Source vs Open Data - what does Tim Orielly think is the ultimate destination for computing? ~23:00
- Where is the money made in open source - software or data? ~ 34:00
- What prediction did Tim Oreilly make in this podcast (2009) that is now coming true? ~51:32
- radar.oreilly.com What is the lag time from articles on this site to the main stream media? ~55:00

2.3.3 - Lab

This activity can be induvidual or group based.

There was a documentary movie called Revolution OS - https://www.youtube.com/watch?v=jw8K460vx1c made in 2001. Answer the questions listed under "Review Questions" above.

3 Hardware and Installation

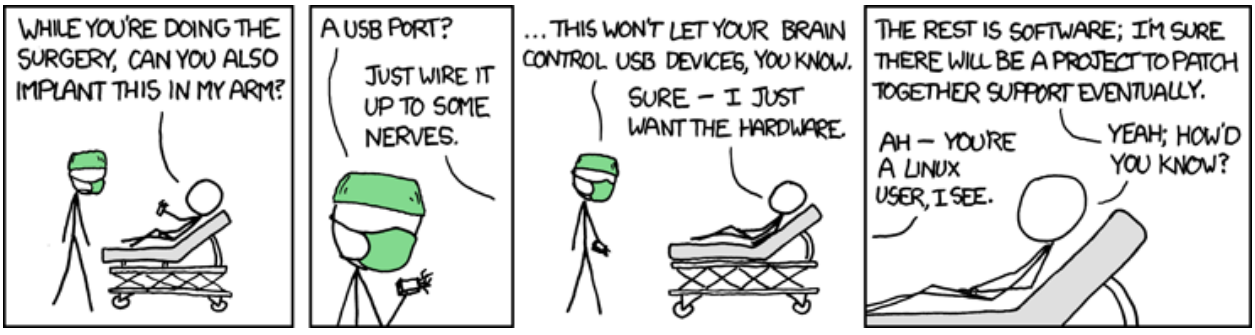

Figure 10: *Must be a Linux User...*

Chapter 3 Objectives

- Know how the Linux install process works for the two major Linux Distribution families
- Know how filesystem partitioning works
- Understand how to use virtualization platforms for installing Linux distributions
- Be able to understand the benefits of automated answer files for installs

Outcomes

At the end of this chapter you will understand the Linux installation process and be able to describe the process fully. A user will also b e familier with the different processor architectures. You will be aaware of virtualization products and platforms the Linux can be installed upon.

3.1 Installation of Linux Distros

Part of the power of Linux is that it is *"free to use"*, *"free as in freedom"* This usually translate into free to use *"cost wise"* as well. This makes the barrier to entry in using a Linux distro very small. In the interceeding 21 years the various distros have perfected packaging and installtions has become very simple. If you are familiar with the Windows or Mac install process then Linux will not be too different. FreeBSD on the other hand, you will find completely alien but that is another story. The term for a file used to install a Linux distro is called an *iso*. An .iso file is actually a standard file type that represents the contents of a CD/DVD-ROM in a single archived file format. Since it is a standard ISO files can be mounted with in operating systems, the can be read from and even written to CD/DVDs, USB, and SD cards. THe reason the iso term and format are so tied to Linux historical one. During the mid 90's as Linux rose to prominence CD-ROM drives and technologies began to become standard amongst PCs. It made sense to create distributions that were almost the exact size of a CD-ROM beccause it made distribution and copying very easy. In the early days of Linux it was not uncommon for a distribution to have a mailing address where you could write a letter and request premade CD versions of a distro.

As USB drives and SD Cards have surpassed Optical Disks in speed and capacity they have come now to represent the favorite install media. Infact if you think about it many laptops, 2 in 1s, and even desktop PCs and Macs don't even come with an optical drive. Though many old an still usable PCs still have optical drives. One of the best tools I have found for creating bootable install media on optical disk or UBS drive is UNetBootin. The tools takes the difficulty out of making install media. It even includes and option to manually just-in-time download which ever iso you are looking for and "burn" the iso to the drive. You may hear the term *"burn"* used in relations to isos, all this means is to transfer or write data from one source to its final source.

ISO files also have utitility for when you are installing a Linux distribution into a virtualized platform.

3.2 - Virtual Machines

Every operating system has the concept of rings in relation to how systems communicate. These rings are for privilege seperation and how security is built in to an operating system. With the higher numbered rings be the least privilleged. Traditionally user applications are in ring 4 and the kernel which has the most power is in ring 0. For instance a program a user write cannot just talk directly to the video card and write to the screen. The program needs to go through the OS which in turn goes through the kernel allowing or enforcing commands to be executed. A hypervisor is a new ring that inserts itself between the OS and the kernel–called ring -1.

"To assist virtualization, VT and Pacifica insert a new privilege level beneath Ring 0. Both add nine new machine code instructions that only work at"Ring -1," intended to be used by the hypervisor." Andy Dorman - Informationweek

When dealing with virtualization you are functionally running multiple operating systems at one time. Technically this is not possible as only one oeprating sysetm can have control at a time - so how can a hyper-visor make this work?

Insert screen shot of system running Windows and also with Ubuntu and Fedora up and running with task manager

By having the hypervisor intercepting system calls from the virtualzied operating system. The way a hypervisor works is not unlike having a professional translator at a business meeting translating between two

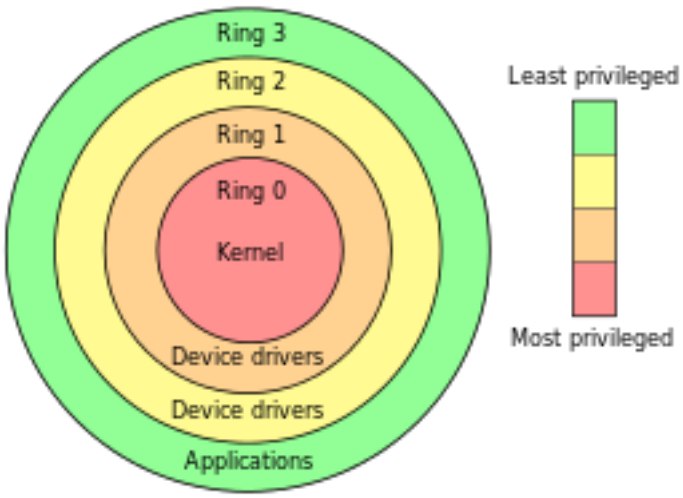

Figure 11: Privilege Rings

atendees. The *"guest"* operating system thinks it has complete control of the hardware - the virtualization software is only showing the guest system a small portion of all the total RAM, CPU, and disk space available. The hypervisor ina sense offers a pretend kernel to the guest virtualized system. In turn, the hypervisor translates the system commands to the kernel it has received and translates them to the host operating systems commands. For example if we are running and Ubuntu Desktop virtualized guest system on a Windows 10 host, the Linux desktop has no way of knowing how to issue a command to use the network card to request a web site because Linux knows its own OS and kernel and Windows is a completely different kernel and operating system. The virtualziation layer will do this translation for you – allowing the *"host"* system to think that your guest virtualzied OS is nothing more than an application, and allowing your guest virtualized operating system to think that it own the entire set of hardware.

The entire concept of virtualization is too large to cover here. I will try to give you a basic introduction. The main concept you need to know is that your computer (PC, laptop, Mac) have vastly more power then needed most of the time. In general you are only actively using your memory, hard drive, and CPU a small fraction of the time. Even watching a Youtube video or listening to streaming music service doesn't usually tax your system that much. Virtualization adds the concept of a hypervisor. A hypervisor is a shim that is inserted into your operating sysetm to intercept system calls to hardware. The benefit is that a hypervisor can act as a translator for multiple operating systems running simultaniously on one system. There by maximizing the usage of yoru resources and preventing you from needing 4 or 5 different physical PCs.

TYPE II

One is geared assuming you are using an underlying operating systems - such as Windows, Mac, or a Linux distro - thereby off loading all control of the programs to the host OS. This is usually called desktop virtualization.

There are four main desktop platforms for virtualization:

Microsoft Hyper-V

Hyper-V was originally only a server class product release in Windows Server 2012 R2. Microsoft ported the technology to be able to be used on Windows 8 Professional and Enterprise as well as Windows 10 Professional and Enterprise. It comes as a free component and is a fully functional implementation of the sever class prodcut. It has the added benefit of being able to work over a wireless connection too. Hyper-V is a good product, you can install Windows and Linux virtual machines on it, but Hyper-V can only run on Windows.

Oracle VirtualBox

This prodcut was originally an opensource project that was purchased by Sun and then inherited by Oracle. Though the name is on the project, Oracle has been surprisingly hands off of this project. Because of that it has grown in usage, features, and utility to become the defacto desktop virtualization tool. It can run on Mac, Windows, and Linux and allows for seemless transfer of virtual machines across platforms

VMware Workstation

VMware also released a desktop product that is similar to VirtualBox on Windows and Linux called VMware Workstation. This software predated Virtual Box by nearly 5 yearswith a seperate desktop product availalbe for the Mac called VMware Fusion.

Parallels Desktop for Mac

Until 2013 Parallels Desktop was a direct competitor to VMware Workstation of the desktop of Windows and Linux. As of 2013 those products were discontinued in favor of Parallels focusing their desktop product on the Mac.

TYPE I

The other is what is called a bare metal hypervisor. These are usually used in server environments on hardware utilizing multiple core CPUs, mutli-terraBytes of RAM, and multi terrabytes of Hard drive space. This Hypervisor includes a kernel and mini-operating system tuned just for managing and interfacing with virtual machines. This book will not cover TYPE I hypervisors or commercial implementations of them.

- Microsoft Hyper-V
- VMware ESXi

3.3 - Installations and isos

Now that we understand a bit about what a hypervisor is let us beging the process. The next pages are going to show you in comparison how to install the latest versions of Fedora and Ubuntu (as of August 10th 2015) Fedora 22 and Ubuntu 15.04. This will require you to download two isos from their respective download sites. For this install process we will assume that you are using VirtualBox version 5.0.0. These distributions can be install directly to a hard drive and become the primary operating system. This might be a good exercise if you have an old laptop or PC laying around. You would be suprised if you asked your relatives or perhaps a company you work for or even a school you might go to what they have in the way of old computers that might still be useful to experiement with Linux installations as well. There is also the concept of dual booting your PCs with multiple operating systems. I created a quad-boot system containing Ubuntu, Fedora, Centos, and Windows 10 see article here for how to accomplish this task. This processes is beyong the scope of this book but link is provided for those interested.

You also need to be aware of the type of architecture you are installing to. Is the processor 32 bit or 64 bit? If it is a 64-bit processor you can install 32-bit operating system but not the other way around. 32-bit processors are usually only low end older Atom processors and older intel chips - pre Core 2 Duo processors. You can find information about your processor by going to http://ark.intel.com. This is Intel's clearing house for all its information about processors and motherboards. They can tell you all you want to know about a processor. All but the most secialized or low end chip these days is 64-bit you should be safe with that type of distro.

The 32-bit distro is most commonly referred to as the x86 or 586, 686 architecture. The 64-bit architecture is usually referred to as x64, but sometimes x86_64, and even AMD_64 <– that is not a reference to AMD processors - just a credit in the name as AMD was the first company to implement 64 bit extententions to the x86 instruction set and they caught on and stuck–hence the credit. Their is one other type of architecture called ARM. This is noted as aarch and aarch_64 – ARM is the architecture that runs phones, tablets, and small embedded sytems such as the Raspberry Pi. It has an entirely different instruction set so the software compiled for this architecture and vis versa is not compatable with the Intel x86 x64 architecture.

Each distro also has a checksum feature provided by the site that issues the download. A checksum is a 1 way mathematical function that gives you a unique representation of what the content of the iso should be. That way if you download and iso from somewhere and the checksum is different then you might be alerted to someone trying to add additonal contents to an iso or perhaps just a corrupted download. Most distros use a simple md5 hash, but a few, notably Fedora, has moved to more robust hashes using SHA-256 strength hashing.

Links to get you started

- Get Fodora
  - Fedora checksum page for Linux and Mac
  - Fedora checksum page for Windows
- Get Ubuntu
  - Ubuntu checksum page
  - Microsoft Powershell hash checking functions
- Get VirtualBox

Here are the commands to execute in Windows in powershell

```
Get-FileHash .\ubuntu-15.04-desktop-amd64.iso -Algorithm MD5 | format-list

Algorithm : MD5
Hash      : 53C869EBA8686007239A650D903847FD
Path      : C:\Users\palad\Downloads\isos\ubuntu-15.04-desktop-amd64.iso

Get-FileHash .\Fedora-Live-Workstation-x86_64-22-3.iso -Algorithm SHA256 | format-list

Algorithm : SHA256
Hash      : 615ABFC89709A46A078DD1D39638019AA66F62B0FF8325334F1AF100551BB6CF
Path      : C:\Users\palad\Downloads\isos\Fedora-Live-Workstation-x86_64-22-3.iso
```

Here are the commands executed in Linux

3.3.1 - Planning Your Install

Before beginning there are a series of questions you should ask yourself, "What do I need in this distro?"

- Strict Security such as SE Linux?
- Stable release with long term support?
- Will this be a desktop install or server install? GUI or no GUI?
- What software will you be needing?
  - Serving web pages?
  - Building Android applications?
  - Hacking your neighbor's wi-fi?
- What processor do I have, 32-bit or 64-bit? How much RAM do I have or need?
- Is this an old PC or laptop I am using–does it lack processor extensions that can aid in rendering multi-media efficiently?
  - SSE
  - AVX
- Licensing for business?
  - Does it need to be GPL compliant?
  - Can propriatary programs and codecs be included?

3.3.2 - VirtualBox configurtion

If you are using Windows, Mac, or Linux you need to download the appropriate version from the VirtualBox homepage. Version 5.0.2 (August 18th, 2015) is the current version.

Feature List for VirtualBox

- Guest multiprocessing (SMP).
- USB device support.
- Seamless windowing
- Shared folders
- Hardware compatibility.
- Full ACPI support.
- Multiscreen resolutions.
- Built-in iSCSI support.
- PXE Network boot.
- Remote machine display
- Video and screenshot capture within virtual machines

3.3.3 - Create Guest Virtual Machine

Upon completion of a fresh install and luanching of VirtualBox you should see this image:

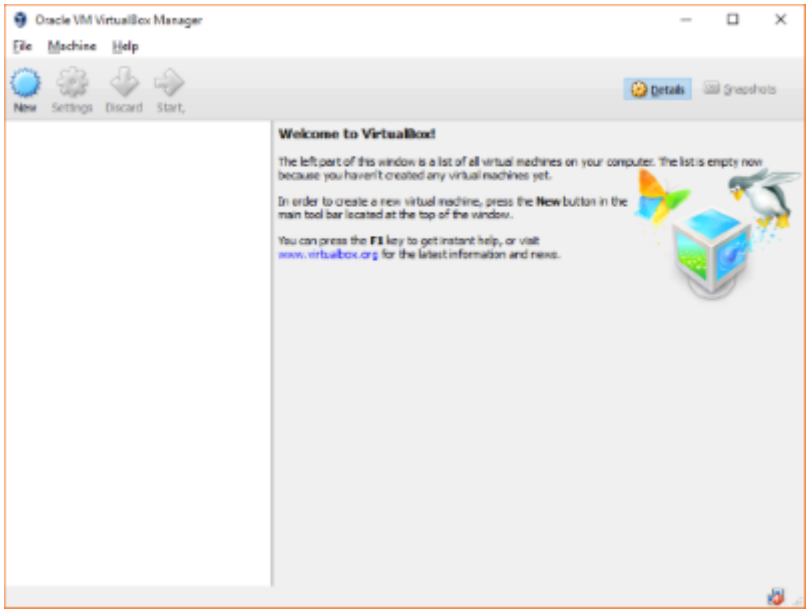

Figure 12: VirutalBox fresh install

See the getting started manual for a wide range of information. Unlike some opensource projects this doucmentation is actually very thurogh and useful. VirtualBox has a list of supported host operating systems, which is basically any operating system you can think of from DOS to Haiku to FreeBSD.

All things start with the NEW button shown in the picture below.

Name it (auto guess) choose the specific Operating System type and then the specific OS. What happens if you choose the wrong one? Two things, 1 you can always go back in the settings option and change it after the virtual machine is powered off. 2. As long as you have the correct OS family everything should be ok–but see the extensions setting at the end of this chapter.

Next is the amount of memeory availalbe - note that this is shared with your underlying OS and whatever you allocate to this guest VM will be unavaialbe to the underlying host OS while the guest VM is powered on. Here you have a choice of how much harddrive space you will allocate to the guest VM. This space will be treated as a file by the underlying host OS–allowing for easy migration, export, and even cloning of the guest VM.

You can chose to dynamically allocate your harddrive space or statically allocate it. The advantage of dynamically allocating is that not all the space will be assigned right away. The harddrive will grow incrementally as you need space until it hits the maximum you defined. The disadvantage of this is that if you are creating lots of data there will be overhead processing in continually allocating enough space. Statically allocating the harddrive space on the otherhand will potentially lessen the number of systems that can go on your harddrive because potentially much space that is allocated is actually unused.

Next is the harddrive file format. There are a few competing standards. If you know you are going to be working in the VirtualBox environment the default VDI is sufficent. If you know you will be transferring this VirtualMachine to another environment: VMWare (VMDK), Microsoft Hyper-V (VHD), KVM (QCOW,QCOW2, RAW) then you can choose the appropriate type.

Now click finish and you should be ready to go - with your VirtualBox start screen looking something like mine. Note I have gone through and completed the setup for both Ubuntu 15.04 and Fedora 22 Workstation as seen here.

3.3.4 - Walk Through the Settings

Before we hit that start button - lets select one of our virtual machines and take a look at the content of the SETTINGS button. Here we will find all the settings possible related to our virtual machine. Though not entirely correct - you could think of this similar to a BIOS settings on a PC - and area where we can configure any underlying hardware.

Define settings for new install

Ready to begin

Video and screenshots Links Here

3.3.5 - Hard Drives and Partitioning

Start the Linux install from iso file

Formatting questions

LVM (see later chapter 13)

Video and screenshots Links Here

3.3.6 - Installing software

The final process where it asks about additional software to install

Video and screenshots Links Here

3.3.7 - VirtualBox extensions

You may have noticed that when a guest VM is usccesfully installed the screen resolution maybe very small or the mouse intergration features are not working. VirtualBox guest additions also enable exclusive features that are not normally availalber in an operating system such as shared folders, cut and paste support, and

even support for multiple monitors. The way to solve this is through something called VirtualBox Guest Additions. On the VirtualBox menu under INSERT DEVICES you need to select the "Insert Guest Additions CD Image."
This is source code and drivers provided by VirtualBox that will add VirtualBox features and hardware to the underlying guest VM. The guest VM has no idea it is in a virtualized environment and even if it did know Oeprating Systems do no come equipped with drivers to support VIrtualBox. In this way the drivers are added to the OS in Windows and in Linux the drivers are loaded into the kernel to enhance the experience.

For Windows and Mac as the guest VM OS this is a pretty stright forward install - the attached Guest Additions iso appears within the VM and you simply double click it and run through the menu, reboot, and new features are added.

For Linux you need to compile these extensions into the kernel and some extra tools are needed.

Debian/Ubuntu

This goes for any distro that derives from Debian. You will need to install the following from the commandline to assist with the compiling and installation of VirtualBox drivers. Apt is the Debian/Ubuntu package manager, we will learn about more in depth in chapter 10

```
sudo apt-get update
sudo apt-get install build-essential dkms linux-headers-$(uname -r)
cd /media/cdrom/VBOXGUESTADDITIONS_5.0.2_102096/
sudo VBOXLinuxAdditions.run
```

Red Hat

On Fedora 22 using dnf

```
sudo dnf update kernel*
sudo reboot
sudo mkdir /media/VirtualBoxGuestAdditions
sudo mount -r /dev/cdrom /media/VirtualBoxGuestAdditions
sudo dnf install -y gcc gcc-c++ kernel-devel kernel-headers dkms make bzip2 perl
cd /media/VirtualBoxGuestAdditions
# 32-bit and 64-bit systems run following
./VBoxLinuxAdditions.run
sudo reboot
```

On Centos, RHEL, and older Fedora distros using yum

```
sudo yum update kernel*
sudo yum install gcc kernel-devel kernel-headers dkms make bzip2 perl
sudo mkdir /media/VirtualBoxGuestAdditions
sudo mount -r /dev/cdrom /media/VirtualBoxGuestAdditions
cd /media/VirtualBoxGuestAdditions
# 32-bit and 64-bit systems run following
./VBoxLinuxAdditions.run
sudo reboot
```

If successful you can reboot the Linux guest VM and you will notice the changes take place immediately. without these additional tools installed you will receive an error message similar to

```
Building the main Guest Additions Modile[Failed]
```

3.3.8 - Automating the Install Answer Process With Preseed and Kickstart

All the previous steps took maybe 10 to 15 minutes if you are on a fast machine which is not bad at all. But let us say you will be creating many virtual machines for research purposes. Or perhaps you will be recreating the same virtual machine many times. There is a way to automate the install process. This is called an answer file in the Windows server world. For Red Hat based systems this is called kickstart and Debian and Ubuntu use a file format called preseed. None of these formats are compatible with each other but there has been some work to get limited kickstart support for Ubuntu.

Debian/Ubuntu pressed template

Kickstart documentation - it can be generated from scratch or upon a succesful install a default kickstart is located in /root/anaconda-ks.cfg

Link to video on how to run the files

screen shot showing you need to host the file on the web somewhere or include the file in install media

3.4 - Chapter Conclusions and Review

Through this chapter we gained an understanding of what x86 based virtualizations does. We learned about the purpose of a hypervisor and how opensource tools such as VirtualBox provide these services. We learned how to install Ubuntu and Fedora based distros in the most common scenarios. We learned about VirtualBox features and how to automate Linux installs through kickstart and preseed configuration files.

3.4.1 - Review Questions

- Questions go here

3.4.2 - Podcast Questions

Listen to the FLOSS podcast number 88 with Linus Torvalds - http://twit.tv/show/floss-weekly/88

- ~6:32 Who is Linus Torvalds?
- ~6:54 Where did he create Linux?
- ~7:30 What did Unix have that other operating systems didn't at that time?
- ~10:02 Within a few months of Linux first release roughly how many people were interested in Linux?
- ~10:30 About what month and what year did this happen?
- ~10:40-13:30 What was the initial inspiration to create the Linux Kernal as an open source project?
- ~13:30-14:00 Why was it licensed under the GPL license?
- ~20:48 Why didn't Linus want to work for a Linux company?
- ~41:00 More than the technology hurdle what else is needed to get into Linux Kernel Development?
- ~46:10 What is the way to become a great programmer?
- ~51:17 What is Linus' farewell message to the audience?

3.4.3 - Lab

- You will need to some research and find the download links for the Linux and BSD based distors below and install them in VirtualBox. Complete the install and launch a text editor and type your name, the name of the Linux distro, and the message "Hello World." Assume each instance listed below is 64-bit version.
  - Debian Based

    * Ubuntu 15.04 Desktop edition
    * Linux Mint 17
    * Lubuntu 15.04 Desktop edition
    * gNewSense 3.1

    * Debian 8
  - Red Hat Based
    * Fedora 21
    * Fedora 22
    * Centos 7
    * One Fedora spin of your choice
  - BSD based
    * FreeBSD 10.2
    * OpenBSD 5.7

4 Desktop Linux (GUI)

Figure 13: *Who needs Flash?*

4.1 From Paper Tape to CLI to GUIs to 4K

Chapter 4 Objectives

Outcomes

Unix had its starts in the late 1960's and 1970's. Computing at that time took on a less interactive and more iterative/batch processing style. Not until the late 70's and early 1980's do we begin to see the green colorede terminals we are familiar with. Initially Unix had to develop support for GUIs. This was done through the X11 protocol or what ended up being called X Windows. The X was originally a place holder in the hopeps that an actual name could be created, but they say what is temprorary is permanent, and X Windows was here to stay. X Windows is actually a client server protocol. It was designed with the idea in mind of transporting display windows over TCP/IP. Seeing as it was designed in the University and corporate setting - the idea of security was non-existant. Any Computer that could connect to another computer could start an X Windows session. The utility of this was that you could use lesser hardware and simply via the X Windows protocol remotely run you application window. An office or university would have 1 large server and each desktop would connect and tunnel X Windows over TCP/IP.

Initially this was designed witha single purpose, single window in mind. But X Window use grew the utility of X Windows to be a *"compositor"* for drawing a GUI (Graphical User Interface) on your local desktop became a standard feature.

X Windows was developed by MIT and someone else. The actual protocol that data transfers over is called X11.

X has a long history and that history was forked and rejoined over time.

X Windows protocol was forked in X86free.org and X.org and for many years were seperate - but they have recently joined back together and the current and standad Linux desktop compositor is now referred to as *"X"*. X has a definate advantage in that it is very mature and very well known for all its good and bad parts. In the diagram below you can see one of X major faults. (Get picture from Wayland website) Seeing as it was designed not with a desktop GUI in mind, every desktop element is a client that has to make calls to the X server in order to render any changes to the screen. This adds extra layers of overhead.

A project was started to reimainge the nature of the Linux compsitor. This project is called Wayland. But the Wayland authors knew they couldn't just throw X out as everysingle Linux desktop today uses it–that would break all of Linux. Instead they reimplemented X into something called X wayland (see image from Wayland.org) Slowly many Linux distributions are looking at the improvememnts of Wayland and moving to gradually implement it. Will we ever see the end of X? Probably not as it is too deep into the "bones" of Linux. But Fedora 22 has a beta version of X-wayland running Fedora 22 - which will render Walyand and X at the same time.

Not ot be outdone Ubuntu decided not to work with the Wayland project and decided to create their own compositor called Mir. THis idea was highly ciriticzed (find link). But I think it was without warrent. Ubuntu has a business case–they were looking to make a compositor that could adapt based on form factor. Instead of having a phone, tablet, tv, desktop rendered, Mir would be custimoizable to Ubuntu's hardware dreams. This precluded them to work on their own compositor. But they have leanred that it is not so easy and it has been constantly delayed from public release. (Link to Mir current status)

4.2 Window Managers

With the advent of X and windowing capabilities, a need to manage multiple local windows arrose and gave rise to early windowing managers. For those of us old enough to remember, this would be similar to early Mac and pre Windows-95 operating systems. (Show image of windowing tool kit)

Window Managers are very fast because they have almost no chrome or what we would think of as standard polish. There only concept is render contents inside of particiular windows. Things we come to expect such as sliders, close boxes, touch simply don't exist in Window Managers

A Window Manager is one step above a single X WIndow extension, but it is not a desktop environemnt. Here is a list of some Window Managers

- Enlightenment
- I3
- Xmonad
- OpenBox

4.3 Desktop Environments

By the time of the Linux kernel release and it becoming a stable development platform. What enabled Linux to move from hobby OS to real commcerial option was the creation of the first desktop environments. The KDE (K Desktop Environment) was the first toolkit released In 1994/5. It was opensources and adopted quickly but parts of it used the propriatery QT (pronounced cuteie) Windowing tool kit - which was not opensource. It was opensourced by KDE 2.0. This moved Miguel De Icazza to create a truly opensoruced alternative to KDE called GNOME. This was a GNU project and also included the development of the GTK Gnome Took Kit for creating windowing objects.

Miguel ended forming the company that became Xamarin, a cross development movile platform using c# to develop for Android and iOS. (Get picture of Miguel)

Which is better? Hard to say. Both have had set backs and advancements over the years. The look and feel of KDE resembles traditional Windows as it was designed to help ease of Windows users into Linux transition. GNOME itstead went for the Mac route of floating windows. Not to be outdone. Ubuntu introduced their own Desktop Environment called Unity (date and time and link to Ubuntu) This decision lead to a many Ubuntu based distros being formed just to replace the Desktop Environment. The majority of Linux distros use GNOME as their standard desktop0 environment. But as in Linux, you can customize or even replace or install side by side the Desktop evnrionemnt. A desktop environment needed to embody more than just window openings and closings, but began to provide tools you and I take for granted. Things such as a clock, or a text editor, office sutie or email client, even initially a web broweser and all these things having a consitent usage pattern and feel.

Seeing as KDE and GNOME focused on features and usability, many people who were using older hardware felt left out or unable to run these Environemnts as the resources required were growing. So a movement to create light desktop environements sprung up. The first was XFCE and then LXDE (How is it related to LXQT) There were also design revolts. When GNOME moved from version 2 to version 3 the amount change was seen by some GNOME users as treason. They foked the GNOME2 desktop code and it became known as something called MATE - which was integrated into a Desktop environment called Cinnamon. All of these desktop environments are available for install. Some are specifically packaged by Red Hat and Ubuntu to match a theme and style and some are avaialble to install but might not be in the most usable state.

Get logos for these

- Desktop Environments
  - KDE
  - GNOME
  - Unity
  - XFCE
  - LXDE & LXQT
  - MATE (GNOME 2) Cinnamon
  - http://smashingweb.info/mac-os-x-theme-for-ubuntu-14-04-macbuntu-transformation-pack/

4.4 - Chapter Conclusions and Review

Conclusion goes here

4.4.1 - Review Questions

- Questions go here

4.4.2 - Podcast Questions

- Best short interview I ever heard with Richard Stallmand from Will Backmand of BSDTalk
- Link to .ogg audio interview file

4.4.3 - Lab

- Using the virtual machines you installed in the previous chapter, you will now install the list below of Window Managers and Desktop environments. You will take a screenshot from within VirtualBox.

5 Linux basic commands and File system structure

Figure 14: Understanding the Technology and Philosophy of Unix/Linux

The main use and power of an operating system to this day is still the Shell. The shell is a way for a user to directly interface with the operating system. We will cover more about specific shells and features in Chapter 8.

Chapter 5 Objectives

- Understand the function of the Linux Shell and its relation to the Operating System.
- Learn how to read the structure of commands on the commandline
- Learn the Linux commandline nomenclature
- Understand the Linux Standard Base and what makes a distro a Linux distro
- Understand the difference between absolute and relative paths
- Know basic tools for moving and modifying the contents of the filesystem.

Outcomes

At the completion of this chapter you will understand how to use the Linux shell for modifying the contents of the operating system. You will understand the nature of the filesystem and how to navigate it–understadning the system path. You will know basic commands for manipulating content in the filesystem.

5.1 Shell

Show picture of seashells

(insert diagram about shell OS interaction)

Many may say, *"Hey I have a nice point and click GUI why do I need to use the crusty old commandline?"* That is a valid question. In reality the GUI is syntactic sugar on top of the Shell. Anything you click on in a GUI in reality is executing a command in the shell. In certain cases using the shell may have more avaialble features for your command then in the GUI. The GUI by definition cannot have more capability than the shell.

In the shell is where we can enter basic commands for navigation and file manipulation. Some of the basic commands we will cover are as follows:

- `cd - used to change directory`
- `ls - used to list the content of a directory`
- `cp - used to cp the contents of a file, can also be used to copy and rename a file`
- `mv - used to rename a file in place`
- `mkdir - used to create or make a new directory`
- `touch - used to create a new blank file or to update the timestamp of an existing file without opening it`
- `cat - technically used to concatenate the contents of two files, but will accept nothing as the second paramter thereby just being used to display the content of a file`
- `less - used for paging the contents of a large file, also supports scrolling up as well as down`
- `date - used for outputting the current date and time in various customizable formats`
- `man - the manual command used to find out how to use the detailed structure of a command`
- `pwd - used to print out your present working directory--your location in the filesystem tree.`
- `file - used to find out what the content type a file is`

5.2 Basic Commands

(show screen shot of ls command with ls ls -la ls -la /etc/ man ls

There is a common nomenclature of commands in Linux. There is an executable that is part of the system function located in /bin, files such as ls cd touch are all precompiled binaries located on the system. To enter

a command you type the name of the binary, as you use Linux more and more you will begin to memorize the tool names. Each command can have options or sometimes called flags and then may or may not accept arguments.

5.2.1 - Command nomencvlature

```
ls -la /etc
```

The first two letters **ls** make up the command for listing the contents of a directory. The command must be followed by a space. Then next letters are preceeded by a **dash**, to tell the shell interpreter that these letters are options. Options are usually single letter representations of functionality. The **-l** options tells the ls command to give a long listing of a directory with details and the **"-a"** tells the shell to print all files in the directory including hidden files. Options can be combined in most cases into a single string preceeded bya dash. So **-la** can also be writted as **-l -a**. Additonally there are options that use full english lanugage structure, which are usually preceeded by **two dashes** and then a more descriptive english phrase. Ask the students to find one?

The final value of **/etc** in the command **ls -la /etc** is an argument passed to the **ls** command telling the ls command to list the contents of the _**/etc** directory. If this value is left empty the shell assumes you mean the **pwd** or your current location/.

5.2.2 - Manual (man) command - your best friend

Purpose of manual command

___history of maunal command

5.2.3 - File System structure

What is a Filesystem?

A way for the Opertaing system to access and manage files
Upside down tree Top of the tree / (root)

Add what each of these locations holds - have the students cd into these directories and do ls and file commands

- /etc
- /bin
- /sbin
- /tmp
- /var
- /usr
- /mnt

talk about Red Hat Explain the sym-linking of /usr/sbin and /usr/bin

optional non-standard /opt from Unix /media Ubuntu and Red Hat - an obvious place to put mounted USB, CD-ROm and other added devices

5.2.4 - POSIX and Linux Standard Base

POSIX

Need for POSIX standard and what it does, and almost sacredness of Linux supporting POSIX standards

Quote from Poettering about dropping the fantasy UNIX and breaking POSIX to enhance Linux capabilities.

LSB

Similar to POSIX but additionally unique to Linux in defining what a distro needs to be officially called a Linux distro.

5.3 Explanation of PATH

```
 * Absolute vs Relative
 * Go over these concepts with above simple commands
```

5.4 - 3 P's

Path Permission dePendencies

5.5 - Chapter Conclusions and Review

Conclusion goes here

5.5.1 - Review Questions

- Questions go here

5.5.2 - Podcast Questions

- Questions go here

5.5.3 - Lab

- Lab goes here

6 File permissions and ownership

Chapter 6 Objectives

Outcomes

6.1 - Read, Write, Execute

6.1.1 - Tools

chmod chown chgrp ls -la fields

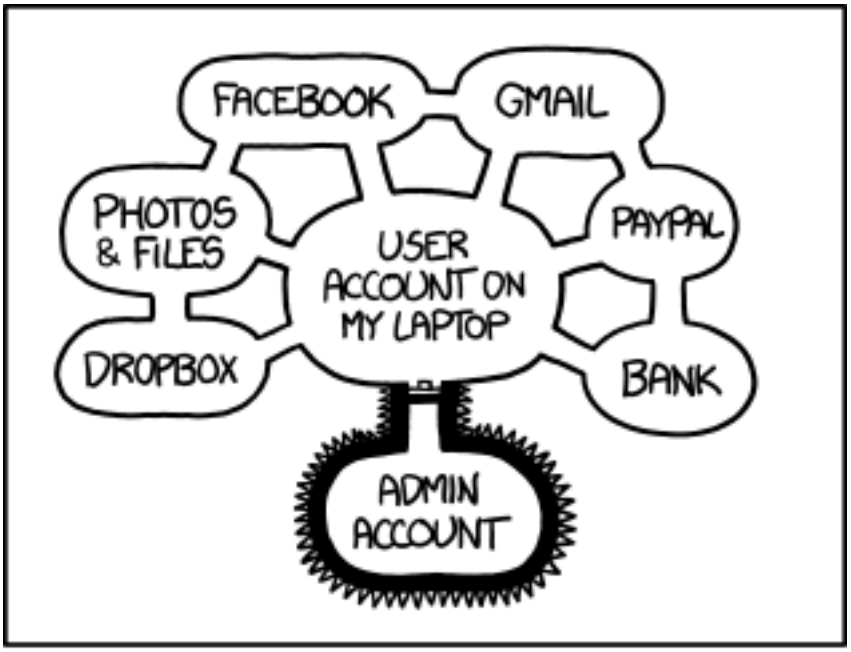

Figure 15: Understanding the Technology and Philosophy of Unix/Linux

6.2 - Chapter Conclusions and Review

Conclusion goes here

6.2.1 - Review Questions

- Questions go here

6.2.2 - Podcast Questions

- Questions go here

6.2.3 - Lab

- Lab goes here # Commandline variables and shell meta- characters

Chapter 7 Objectives

Outcomes

6.3 - Chapter Conclusions and Review

Conclusion goes here

6.3.1 - Review Questions

- Questions go here

6.3.2 - Podcast Questions

- Questions go here

6.3.3 - Lab

- Lab goes here # vi and editors and bash shell and profiles

Chapter 8 Objectives

Outcomes

XKCD vi cartoon goes here

6.4 - History of VI

ed -> em -> ex -> vi -> vim

6.4.1 - Bill Joy and BSD

Get picture of Bill Joy - link to article The future doesn't need you

6.4.2 - Why keybindings are as they are

6.4.3 - Relation of vi and vim

6.4.4 - Stream editors vs text editors

vi and emacs vs nano, gedit, joe, kate

6.5 - Chapter Conclusions and Review

Conclusion goes here

6.5.1 - Review Questions

- Questions go here

6.5.2 - Podcast Questions

- Questions go here

6.5.3 - Lab

- complete vi tutor example # Basic Shell scripting

Chapter 9 Objectives

Outcomes

6.6 types

- if statements
- for loops
- System variables
- Passing Variables into scripts

6.7 - Chapter Conclusions and Review

Conclusion goes here

6.7.1 - Review Questions

- Questions go here

6.7.2 - Podcast Questions

- Questions go here

6.7.3 - Lab

- Lab goes here # Package Installation

Chapter 10 Objectives

Outcomes

6.8 Types

- yum (soon to be dnf replace yum)
- apt-get

6.9 - Chapter Conclusions and Review

Conclusion goes here

6.9.1 - Review Questions

- Questions go here

6.9.2 - Podcast Questions

- Questions go here

6.9.3 - Lab

- Lab goes here # Symlinks and file types

Chapter 11 Objectives

Outcomes

6.10 Types

- Stdin and stdout and stderr

6.11 - Chapter Conclusions and Review

Conclusion goes here

6.11.1 - Review Questions

- Questions go here

6.11.2 - Podcast Questions

- Questions go here

6.11.3 - Lab

- Lab goes here # Find and grep

Chapter 12 Objectives

Outcomes

6.12 Types

- Intro to regex

6.13 - Chapter Conclusions and Review

Conclusion goes here

6.13.1 - Review Questions

- Questions go here

6.13.2 - Podcast Questions

- Questions go here

6.13.3 - Lab

- Lab goes here # Creating, Partitioning, and mounting filesystems

Chapter 13 Objectives

Outcomes

6.14 Types

- Creating them virtually
- fdisk

- df
- mount command
- /etc/fstab

6.15 - Chapter Conclusions and Review

Conclusion goes here

6.15.1 - Review Questions

- Questions go here

6.15.2 - Podcast Questions

- Questions go here

6.15.3 - Lab

- Lab goes here # Basic Networking

Chapter 14 Objectives

Outcomes

6.16 Types

- Network configs
- Tools to troubleshoot

6.17 - Chapter Conclusions and Review

Conclusion goes here

6.17.1 - Review Questions

- Questions go here

6.17.2 - Podcast Questions

- Questions go here

6.17.3 - Lab

- Lab goes here

Services and processes

Chapter 15 Objectives

Outcomes

6.18 Types

- Starting stopping
- Systemd vs sysVInit
- Cron jobs

6.19 - Chapter Conclusions and Review

Conclusion goes here

6.19.1 - Review Questions

- Questions go here

6.19.2 - Podcast Questions

- Questions go here

6.19.3 - Lab

- Lab goes here # Future and embedded Linux

Chapter 16 Objectives

Outcomes

6.20 Types

- IoT
- RaspberryPi
- Android

6.21 Conclusion

- 3 P's trouble shooting (PATH, PERMISSION, dePendencies) Based off of the Army's 3Q's of movement for soldiers – quickly, quietly, and cautiously.
- Linux Plus
- LPI

6.22 - Chapter Conclusions and Review

Conclusion goes here

6.22.1 - Review Questions

- Questions go here

6.22.2 - Podcast Questions

- Questions go here

6.22.3 - Lab

- Lab goes here # Glossary A

hello

7 Glossary B

hello