THE GROWTH OF ANDROID IN EMBEDDED SYSTEMS

THE LINUX FOUNDATION TRAINING PUBLICATION

Written by Benjamin Zores

THE LINUX FOUNDATION

OVERVIEW

Linux has continuously grown in the embedded systems market for over a decade, gaining market share from proprietary operating systems. The proliferation of embedded devices, the explosion of open source development, the inherent hardware support, the incredible networking capabilities and the royalty-free economic model have all helped propel use of the Linux kernel into one of the best choices for the design of new embedded systems.

While the success of Linux in the embedded market can not be denied, its notoriety was once confined to mostly technical professionals. That changed in 2008 with Google's release of the Android mobile phone operating system, based on the Linux kernel. Thus began the tremendous growth of Linux in the consumer world, with over one million Android devices being activated every day in 2012 and predictions of total Android devices shipped reaching one billion in 2013.

THE GROWTH of Android in Embedded Systems

In a recent survey, 34% of embedded engineers are considering using Android in 2013.

The Android Operating System

Android's success was no accident and was the result of a long-term strategy and loads of investment from Google. The early development of the OS came from within Android Inc. in the early 2000's; it was purchased by Google in 2005. The original system relied on a Java framework for its application layer and was not based on the Linux kernel. Only after several years of development at Google labs, and after an architecture revamping, the first Android-based smart-phone (the HTC G1) was released and based on the very first version of the Android software development kit (SDK).

Originally designed only for mobile phones usage, the system has evolved into a variety of new markets, naturally spreading to tablets, set-top-boxes, video game consoles, connected TVs, military communication devices, medical devices and soon to be PCs. Though most consumers are unaware of the underlying presence of the Linux kernel powering their devices, Android has become a household name and it's heavy usage is well known; Android's mobile device marketshare is close to 75% at the time of this writing.

Many device manufacturers are now considering a move to Android as a replacement for proprietary systems from the past such as VxWorks or QNX. In a recent survey by UBM, 34% of embedded engineers reported that they are considering using Android in 2013. For apps-based devices, the Android OS can sound appealing to manufacturers (due to the more than 700,000 applications available in the Google Play store) who would rather not invest in software development, and instead shift their focus to other areas, and potentially decreasing their time-to-market.

Android's native multimedia capabilities also come as a free gift for multimedia-oriented consumer equipment manufacturers. Whether it is for domestic use of set-top boxes (STBs) or for in-vehicle infotainment (IVI) solutions, interest in being powered by Android is rising. Even some high-end residential boiler manufacturers now are proposing Android remote control applications for home automation purpose. The Android operating system is moving ahead quickly and supporting more and more features at each release.

However, what makes life easier for developers through Google's SDK, can present a burden for device manufacturers, who barely manage to follow the release cycle and often have to skip some versions. As pointed out by the figure [1], Android has seen 17 releases in just four years.

NAME	VERSION	SDK RELEASE DATE	KERNEL VERSION	SDK API	NDK API
N.A.	1.0	September 2008	2.6.25	1	N/A
Petit Four	1.1	February 2009	2.6.25	2	N/A
Cupcake	1.5	April 2009	2.6.27	3	1
Donut	1.6	September 2009	2.6.27	4	2
Eclair	2.0	October 2009	2.6.29	5	2
	2.0.1	December 2009	2.6.29	6	2
	2.1	January 2010	2.6.29	7	3
Froyo	2.2	May 2010	2.6.32	8	4
Gingerbread	2.3 - 2.3.2	November 2010	2.6.35	9	5
	2.3.3 - 2.3.7	February 2011	2.6.35	10	5
Honeycomb	3.0	February 2011	2.6.36	11	6
	3.1.x	May 2011	2.6.36	12	6
	3.2.x	June 2011	2.6.36	13	6
Ice Cream Sandwich	4.0 - 4.0.2	October 2011	3.0.1	14	7
	4.0.3 - 4.0.4	December 2011	3.0.1	15	7
Jelly Bean	4.1.1 - 4.1.2	June 2012	3.0.31	16	8
	4.2	November 2012	3.0.31	17	8

Figure [1]: Android's releases and API level of compatibility.

Android comes as a safe solution where anyone can build a custom version without fear of further legal complications.

Life Without GNU

Android is based on the Linux kernel and takes full advantage of the benefits of its incredible hardware support capabilities. However, there ends the resemblance with any other embedded (or even desktop) Linux distribution.

It is important to understand the Android model. Google develops the system and it remains free of charge. While the source is freely available, the development is not done in the same kind of community-way as it is with the Linux kernel.

The Android development procedure is often referred to as “clopen” (i.e. closed-open), as the drop of the source depends on Google’s willingness to share (the Honeycomb release sources never were made public for instance) and comes only after the new version has been announced and advertised. For good or for bad reasons (depending on which side you’re on), Google got rid of (L)GPL licensing wherever possible in favor of Apache/BSD/MIT licenses. Except for the Linux kernel itself, most (if not all) of the 150+ external Free and Open Source Software (FOSS) parts of Android have a non-GPL license. Unfortunately, the GPL has always been the source of many fears within the industry; Android comes as a safe solution where anyone can build a custom version without fear of further legal complications.

For legal, performance and/or convenience reasons, Android’s software architecture differs heavily from traditional embedded Linux distributions. Google has hacked over (or even re-designed) some of the fundamentals in order to fit the appropriate execution context. The Linux kernel itself has been modified to incorporate several “Androidisms.” Google has introduced multiple aggressive power-management features such as Wake Locks and Early Suspend, features which were not popular with mainstream kernel developers and caused extensive debate in the kernel community. One should note that most modern Linux distributions are desktop- or server-centric; unfortunately embedded Linux distributions used by devices consequently derive from that framework.

Targeting mobile (i.e. battery-powered) devices, Android’s power-management policy is designed to minimize power consumption. Thus, the kernel has been configured to go to sleep as soon as possible and applications that need to run must explicitly force the system to stay awake. This is a radically different mode than that used by regular desktop distributions, which historically have tended to mostly follow an “always-on” usage. A few other drastic changes such as the Binder IPC message bus driver, alarm, timed General Purpose Input/Outputs (GPIOs) and paranoid network security features have long been the source of struggle within the mainstream kernel community, but some conciliation and incorporation of features has been done with Linux 3.3, 3.4 and 3.5 kernels.

Android also differs from GNU/Linux by its userspace. The Bionic C library has been designed from scratch, breaking any sort of compatibility with either Glibc or uClibc, and preventing any pre-compiled Linux application from running on Android. The interface layer with hardware and kernel drivers also has been extended by a hardware abstraction layer (HAL), allowing manufacturers to write proprietary closed-source userspace drivers with their hardware, once again probably for legal considerations and to induce wider adoption.

As one can gather from figure 2, the low-level layers have also been stripped out from GNU/Linux. The BusyBox project has been deprecated by Toolbox, the Xorg/Wayland window and composition servers have been replaced by a brand-new SurfaceFlinger, PulseAudio framework’s equivalent is now called AudioFlinger, and GStreamer multimedia framework has been re-invented through StageFright. Needless-to-say, under these

conditions, one has no chance to even try to run a typical DirectFB / GTK / Qt / Enlightenment application as-is.

Last but not least comes the Dalvik VM project. The whole Android application, system and services framework are written in Java. While increasing application portability, Java is not the preferred language of choice of traditional embedded developers. Android, however, is not based on any Java Virtual Machine (JVM). Code is Java, but then compiled and post-processed to generate Dalvik byte-code instead of Java byte-code, as to confer security and performances enhancements to the system. This also led to long-term lawsuit between Oracle and Google regarding property of free use of Java in Android, which finally settled down, as APIs can't be protected by copyright. The whole system framework (along with 2000+ classes) is then loaded only once in memory through Zygote, the mother of all applications. Each new instance then comes as a fork of Zygote memory space (with copy-on-write methods) allowing applications to load quickly while minimizing their memory consumption.

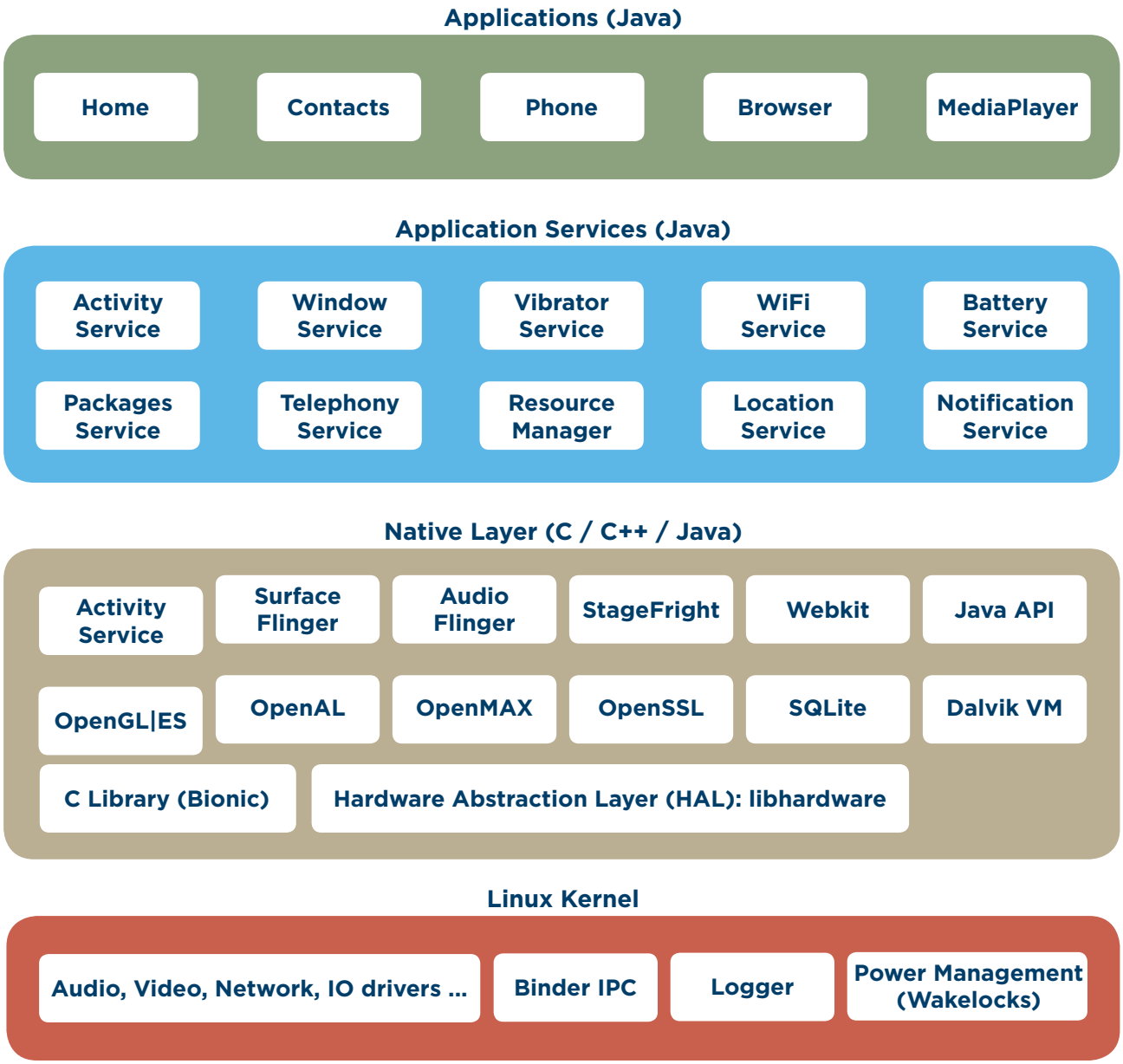

Figure [2]: Android's software architecture.

THE GROWTH of Android in Embedded Systems

For devices that are expected to perform very specific functions, embedded Linux distributions built from the Yocto Project are a much better choice.

Designing and building applications with Android and packaging them once for multiple targets allows companies to drastically save costs and efforts.

Dealing with Embedded Linux Operating Systems

As may happen with any open source software project, Android has been forked by various groups. Any device manufacturer using Android, however, will pick it up directly from Google (perhaps with some hardware support patches extending the base). There is only one Android and Google is actively developing it. There's no need to look elsewhere if your goal is to be compatible with Android.

When it comes to embedded Linux however, the available customizations can come in handy. However, the wide diversity of embedded Linux distribution providers (Windriver, Montavista, Mentor Graphics, etc.) and the amount of DIY open source embedded frameworks (the Yocto Project, OpenEmbedded, Buildroot, LTIB, OpenBricks) and the other SoC manufacturer specific board support packages (BSPs), can make it difficult to know where to start. The strength of being able to select and tune your packages in order to build a perfectly customized distribution can quickly become a nightmare when you have to maintain it and potentially evolve to newer versions. Sure it allows your engineers to cook their own distribution, but companies are investing R&D resources into building and maintaining a system instead of focusing on features and added-value. Android can solve this dilemma by leaving you far fewer choices. Things are the way Google believes they should be, and either you are fine with it, or you go your own way.

Mass advertising has introduced the concept of apps over the past few years. While the idea of applications might seem like a compelling reason to consider using Android, it's not hard to understand that many devices will not get any value from an apps-ecosystem. For devices that are expected to perform very specific functions the way the developer wants them to, embedded Linux distributions built from the Yocto Project are a much better choice.

Reasons for Android's Attraction

Android's success is not a matter of good luck. Its massive adoption by industry is due to several reasons:

- **Rich Application Framework.** Modern GNU/Linux systems offer you complete freedom. One can pick what one likes and use it and modify it at will to perfectly suit one's needs. Forks are common and developers tend to do things their own way, following personal philosophies and beliefs. When it comes to building your own system, one often has many available software bricks, yet none that can be plugged into another as is. One can often miss the big end-to-end picture.

  Android comes with a stable long-term API, provided through an excellent SDK, which brings a somewhat standardized ecosystem and framework to third-party partners, who are able to deal with every part of the system. While the OS is in constant evolution, the API stability across releases mostly remains preserved, allowing you to invest long-term. Designing and building applications and packaging them once for multiple targets allows companies to drastically save costs and efforts.

- **Aggressive Time-to-Market.** Designing devices with Android can significantly reduce time-to-market. Grab the sources, adapt them to your particular piece of hardware and sell it. If you follow reference designs and reference device usages, releasing a new device should be possible in only a matter of months. However, as those who have used Android can tell you, it's far from being that easy and your team should gain some Android-specific expertise and knowledge of the OS internals. While bringing life to your Android device can be faster than what traditional embedded

Google has chosen not to reuse some long developed open source software, leading to some hard-to-fix issues.

Linux distributions can offer, following the system's evolutions and maintaining your code in the long run is yet another story.

- **Focus on "What Really Matters."** By bringing the practical framework to the table, Android allows embedded developers to actually focus on areas that will add commercial value. Board bring-up can be time-consuming enough and one wants to avoid wasting more time on re-inventing yet another embedded distribution on top of it.
- **Open Source.** Though not developed in a community-way, Android remains 100% tunable and provides companies a feeling of safety regarding potential legal threats and licensing.

Under-the-Hood Culprits

While Android has grown to an impressive level of maturity, the OS still has a few demons to cope with, some of them being inherent to its architectural design.

- **Standardization and Economy.** With its massive adoption in the consumer market, Android has become an important standard. The System on Chip (SoC) development costs have grown to such a level of complexity and difficulty of integration that hardware manufacturers can only invest in volume-driven applications and customers.

  Multiple vendors now feature embedded Linux BSP only as an internal sandbox for board bring-up, while delivering Android BSP to their final customers. After all, who would invest in embedded Linux BSP targeting 5% of customers while 95% want Android only? When embedded Linux market shares have decreased, it's been mostly due to vendors not investing as much as they once did. Android's hardware abstraction layer also triggers some limitations inherent to its possibly proprietary licensing. SoC vendors provide their customers the binary HAL blobs meant for a given Android release and platform API. Device manufacturers then depend on their partners' willingness to update and maintain their user-space drivers for each Android version, preventing any further device upgrades and maintenance that is possible otherwise. This is especially true for old, long-running SoCs that could benefit technically from a new Android release, but where no one is willing to invest the work, leaving the end user with no choice but to upgrade to the next generation device.

- **Design Flaws.** For licensing and convenience reasons, Google has chosen not to re-use some long developed open source software, leading to some hard-to-fix issues. One consequence is Android not being real-time capable. Its kernel is often tuned for 1000 Hz low-latency but that is the best one can do. Forget about the PREEMPT_RT patch-set (with proprietary user-space drivers, it would be hard to get anyway) and say hello to the Dalvik virtual machine and its garbage collector.

  While Java application developers are accustomed to not caring about memory management, someone still has to do the job. The Dalvik GC is automatically triggered within application runtime, pausing execution context. Add to the checklist a terrible and slow audio architecture (audio packets come from bottom ALSA up to Java framework through multiple HAL and JNI indirection layers) that prevents any real-time professional audio application to run efficiently.

  Android's multimedia layer isn't in better shape. While it appears to do the job well enough, it is ages away from the Linux industry standards (GStreamer and/or FFmpeg), in terms of performance, architecture, audio/video codecs support and

For many, embedded Linux still remains the operating system of choice but the growth of Android can not be denied.

portability among devices. It is unfortunately not so uncommon for native applications to actually try to guess the hardware they're running on and dlopen() / dlsym() some functions to perform hardware-accelerated operations that are not directly available through StageFright's framework.

Last but not least, Android has deliberately hobbled one of Linux's key assets when developing the networking and connectivity layer. The OS is unable to handle more than one network connection at a time, more than one device driver at a time (per connection type), and unable to handle more than one interface per connection type. Sure, it didn't make much sense in a mobile phone world, but now that Android has spread to markets and places where it was not originally expected to be, problems emerge. From a device manufacturer standpoint, adding simple things like Bluetooth and WiFi, or even basic Ethernet support, is a massive pain compared to what it would take with a more classic embedded Linux distribution. Add on top of that the absence of real routing and access point capabilities, and you'll understand that networking capabilities are quite limited.

- **A trade-off between performance and portability.** As appealing as the Java "write once, run everywhere" framework's philosophy might be, it is also quickly limiting. Any serious performance-critical or multimedia application actually has to be built upon native C/C++ code (sometimes even platform-specific) through Google's NDK, cutting down portability.

- **The limits of "embedded."** To some extent, Android was originally designed for low-power and low-resource devices. The Dalvik VM has been designed to minimize memory use and application footprint, which are precious in embedded devices. Since 2010, newly developed SoC's have come with fantastic performances features.

  At the time of this writing, high-end devices have 4-core Cortex-A9 CPU (with Cortex-A15 just around the corner), 32GB eMMC as storage and up to 2 GB RAM. While becoming more and more powerful, these devices are still a far cry from your current PC, though perfectly manageable for perform your typical daily tasks (i.e. browsing, chatting, email, games, etc.). With Android 4.0 Ice Cream Sandwich, it has become challenging to run without a minimum 512MB memory and an OpenGL|ES capable GPU.

  So, where to put the desktop/embedded frontier? Can we still call such devices "embedded"? How would today's Android releases behave on an old ARM9 or ARM11 device? What if Android has raised hardware requirements just a bit too high?

Conclusion

The growth of Android in embedded usage is not a trend that is up for debate. It has brought the Linux kernel to an incredible number of devices. Today, device manufacturers want to support Android, even if it is just to follow the trend and make sure they are not left behind.

Paradoxically, in some ways, Android somehow has slowed down innovation. Likewise, products all evolve in the same [Google] way, regardless of the manufacturer, with the exception of some necessary custom user-interface rework. It has become more and more difficult to differentiate one device from another, except from the quality of its hardware peripherals. Many consumer-oriented markets (phones, tablets, in-vehicle infotainment) are trending toward multimedia, connected and applications-based devices (where, for some obscure reasons, every big player nowadays wants an application store to monetize its'

Android is not replacing embedded Linux, but instead pushing the Linux kernel in new directions and markets that once were driven by proprietary operating system.

services), and Android clearly can be a winning solution for these needs.

So, will Android be the holy answer to all our needs?

For many, embedded Linux still remains the operating system of choice, especially in areas such as headless product design. While remaining possible, hacking Android to run on headless devices such as routers, probes and sensors, servers and network access points, is nothing but a painful path that only a few brave souls are willing to take. Embedded systems developers also got familiarized with Linux over the last decade and have grown in expertise. Android is a brave new world that requires specialized knowledge.

Alternatively, any product design based around an LCD screen with a touch-capable display, and intending to be apps-driven, should seriously consider Android, even if the product is not a smart-phone. The existing ecosystem, along with the framework's off-the-shelf capabilities, provides developers all the necessary tools at hand to build great products.

Android actually has brought to the market what GNU/Linux misses the most: one single framework that allows application developers to deal with every single part of the system (DirectFB, GTK, Qt, EFL and friends are pretty good toolkits but none of them is as complete as Android's framework). Keep in mind, however, that Android lacks in performance and tuning what traditional embedded Linux does in end-user application design capabilities. As painful as it is to say, today's hardware has become powerful enough to mask Android's performance issues.

So should you go the Android way? The answer is a trade-off to be made by each manufacturer among the orientation of his product and the skills of his teams. If you already have switched to embedded Linux in the past and raised your team to a level of expertise where your code is mature and based on strong software bricks, I'd advise you to keep on this track because Android would require a fresh start. If you're a newcomer, however, (or an old proprietary systems' user) and you missed the last decade's embedded Linux emergence, or if your end product is open and meant to be able to support third-party partners extensions and applications, just listen to nature's call and go with this new industry standard.

In either case, Android is definitely not replacing embedded Linux in the industry, but instead pushing the Linux kernel in some new directions and markets that once were driven by proprietary operating systems. Each track has its pros and cons and while going with Android sounds appealing (and it is!), this kind of decision has to be made wisely, as Android and embedded Linux follow different paths that are not likely to reconnect.

About the Author

As a software architect for Alcatel-Lucent, Benjamin Zores has been designing embedded Linux devices for 10+ years, leading enterprise-grade Linux/Android multimedia IP phones conception. His area of expertise mostly covers low-level devices and platforms definition, board bring-up and drivers development, though his real passion comes from reverse-engineering the software architecture of operating systems to understand what's beneath the hood. He drove the conception of an Android- based wired IP phone and has a very deep knowledge of bringing support for all multimedia peripherals and connectivity layers of Android. Prior from that, Ben was also most known for his open source contributions, as the original author of the OpenBricks embedded cross-build framework, the GeeXboX HTPC live distribution and the uShare UPnP/DLNA MediaServer. Ben is also a recurrent speaker at LinuxFoundation's ELC and ABS events and Android technical writer for Linux Magazine France. He lives in Strasbourg, France.

WHY TRAIN WITH THE LINUX FOUNDATION

We needed someone who could fully engage with Ph.D.-level developers. We had no doubt that we'd found the right instructors.

Dana Krokosky, *Compunetix*

The willingness of the Linux Foundation to customize the course to our needs was the biggest determining factor for choosing them.

Matthew Cheng, *Broadcom*

The Linux Foundation really had the best credibility out there, and they were flexible and tailored the class to what I needed for my developers.

Paul Beer, *Optelian*

Android Training Courses

The Linux Foundation offers several Android development training courses for companies and individuals to quickly get up-to-speed on the Android platform and on writing apps for Android devices:

- Introduction to Embedded Android Development (LF308)
- Introduction to Android (LF329)
- Inside Android: An Intro to Android Internals (LF315)
- Android Bootcamp (LF295)

To learn more about our Android training, visit:

http://go.linuxfoundation.org/android-courses

Distribution-Flexible

The Linux Foundation's courses are built to be distribution-flexible, allowing companies or students to easily use any of the big three distribution families: Debian, Fedora or OpenSUSE. If your company runs one of these Linux distributions and needs an instructor who can speak deeply on it, we have a Linux expert who knows your distribution well and is comfortable using it as the basis for any corporate Linux training. For our open enrollment students who take our online training or classroom training, our goal is to help them, first and foremost, to become Linux professionals, rather than focusing on how to use one particular set of tools.

Technically-Advanced

The Linux Foundation's training program has a clear advantage. As the company that employs Linux founder Linus Torvalds, we are fortunate in our ability to leverage close relationships with many of the top members of the Linux community, including Linux kernel maintainers. This led to the most comprehensive Linux training on the market, delivered through rigorous five-day courses taught by Linux experts who bring their real world experiences to every class.

Since Linux is always evolving, our course materials are regularly refreshed and up-to-date with stable versions of the Linux kernel. We deliver our advanced Linux training in a 50/50 training format, where 50 percent of a student's time is spent learning from an instructor and the other 50 percent doing exercises in hands-on learning labs.

For more information about our Linux training, please visit **training.linuxfoundation.org** and contact us today.