What is this "Linux", anyhow?

Created by Brennen Bearnes

https://learn.adafruit.com/what-is-linux

Last updated on 2023-08-29 02:42:36 PM EDT

Table of Contents

Overview

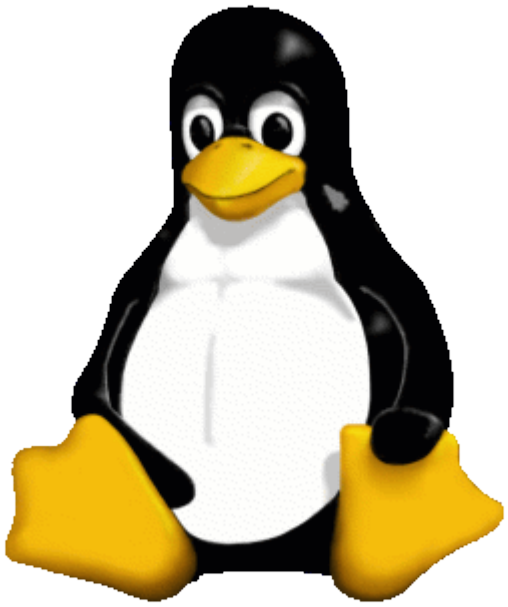
EDITOR'S NOTE: Hiya, Lady Ada here to introduce this tutorial! If you're starting down the path to learning about electronics or computers, you may have noticed or heard about "Linux" - as in "this dev board is linux-based" or "this wearable runs linux" or "I wrote a linux script to control the barcode scanner"

And you might be wondering Well, what is this "Linux" anyhow? Does it matter to me? and then maybe you asked someone and you got a long rant about stuff called kernels and bashed shells and now you're wondering if it's corn-related or is some sort of crab.

Being that this question and confusion is inevitable, and we're getting so many people asking about this mysterious Linux, we at Adafruit thought we'd write up a series of tutorials to help you understand what linux is, when you want linux and how to use it when you do.

This is the first in the series, take it away Brennen!

Linux is an operating system



Tux penguin, the mascot of Linux! ()

Let's begin with the most basic explanation of what Linux is:

Linux is an operating system

Actually, let's amend that:

Linux is a kernel, which put together with software from GNU and BSD and lots of other places, makes up an operating system

OK, great! We're done here, pack up and lets go home!

Just kidding... While those statements are factual, they don't explain much.

So what is an operating system?

An Operating System (we will shorthand it by saying OS) is the software that lets a computer run other software. It bridges the gap between the complicated guts of computers (processors! memory! hard disks! mouse! keyboard! video card!) and programs that need to run on many different kinds of hardware.

For example, it would be really frustrating if you could only run your favorite game on computers with AMD-brand processors and exactly 2 gigabytes of Crucial-brand RAM and only Western Digital SATA-type hard drives. Or if your word processor would only work on a desktop computer and not on a laptop. Since each computer is made of different parts from different manufacturers (which lets you have a lot of flexibility in price, size, speed, etc) we need to have common ground language for talking to all the physical hardware bits that make up your desktop or laptop.

That common ground/language is called the Operating System

Operating systems you may have heard of!

- Microsoft Windows 7 - and its little/big brothers 3.1, 95, NT, 98, XP, Media Edition, 8, 10
- Pocket PC / Microsoft Windows Mobile - the lightweight mobile/phone versions of Windows
- Mac OS X - and previous versions like Mac OS 9, System 7, etc.
- iOS - Apple's mobile/phone/tablet operating system
- Unix, BSD, NeXT, Linux, Solaris, etc. - the huge number of Unix-like operating systems
- There're tons more you can read about on Wikipedia ()

Each one of these was developed by a group or company to talk to hardware in the way that group thought best. In general (but not always) - software written for one Operating System will not run on another operating system. You cannot run a Windows program on iOS (emulators are an exception)

What is in an Operating System?

Modern operating systems contain a lot of software, but they're usually all built around something called a kernel, which is the core piece of code in charge of managing all that hardware, things like processors, memory, hard disk drives, and network interfaces like Ethernet or WiFi.

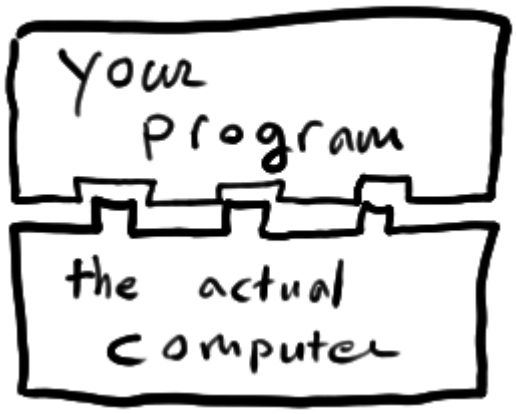
A kernel starts programs and controls the way that they share hardware resources, coordinates driver code for different kinds of hardware, and makes it possible for software (like your spreadsheet) to work without itself knowing the detailed specifics of hardware (like which brand of monitor you are displaying the spreadsheet on).

Kernels vs "Bare Metal"

If you've worked with tools like Arduino or Propeller or PIC or other microprocessors, you've probably written programs that run right on the "bare metal" of a microcontroller. In general, when your code says something like this:

```
// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);             // wait for a second
  digitalWrite(led, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);             // wait for a second
}
```

...then the code in your loop is the only thing running, happily blinking that LED () 'til the end of time (or the end of your battery life, at any rate). You could think of things as fitting together like this:



Other than the Arduino IDE and the compiler, there's nothing in the way of your commands and the computer/microprocessor.

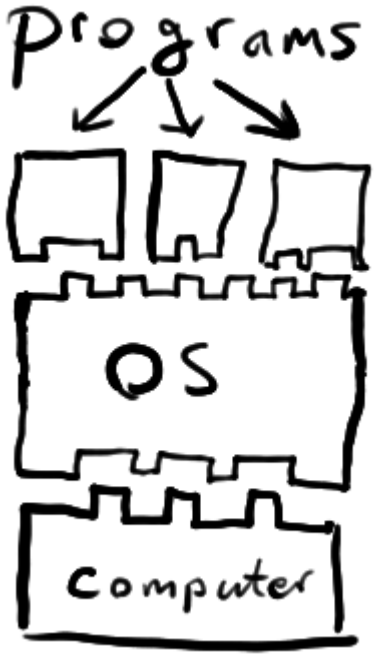
For lots of simple projects, this is a great approach: The hardware is low-cost and runs for ages without much power. The code is simple and can get right down to business without waiting for a lot of complicated stuff to boot up first.

So Why an Operating System?

So what happens when a device needs to do a lot of things at once — stuff like storing lots of data, communicating across complex networks, and interacting with multiple users? What if you want to be able to switch between different programs, or

run the same program on very different kinds of hardware? What if the users of your hardware want to be able to reprogram it on the fly?

These are the problems that led the builders and users of early computers to write operating systems. What they came up with looks a little more like this:

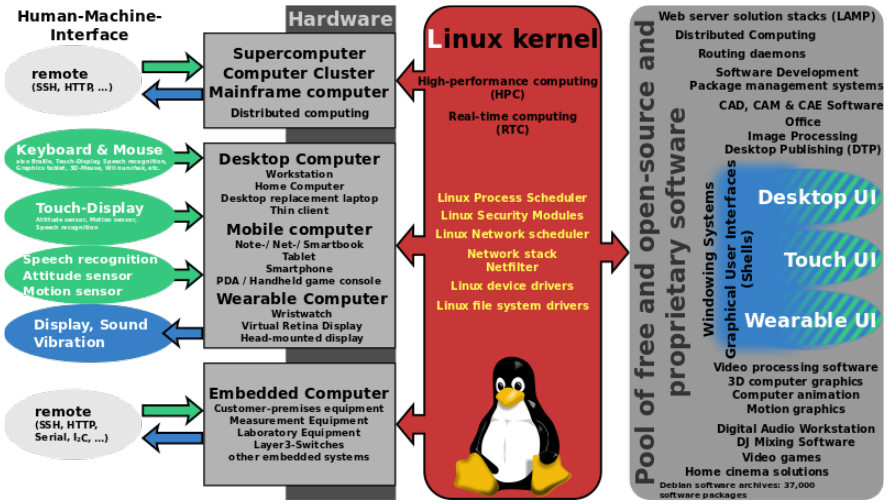


The kernel of a modern OS is a little bit like that loop() function, if you wrote it to enable all of the stuff that you might want to do with a general purpose computer. However, instead of just turning on/off an LED, it goes around to every single program and takes requests for what the program wants.

That turns out to be a lot of stuff.

For example, this morning my kernel asked
"Hey MyPaint, whatcha need?"
and MyPaint replied
"Hey can you draw some red pixels over here on the monitor"
and the kernel commands the video driver to do that.
Then it went over to the IM client and said
"Hey IM client, whatcha need?"
and the IM client said
"Hey I also need you to draw to the montor, this little notice that a new message came in AND also can you play a ding-tone on the speaker?"
and the kernel replied and said "no problem" and told the sound driver and video card to do those things. The more programs running at once, the more the OS/Kernel has to chat to each one, asking what it wants and doing those tasks.

Note that some times, the kernel/operating system cannot handle or do the request! That's when you get program crashes, blue-screens-of-death, error boxes, etc. MyPaint may say "Cool time to save this image file, please save it to disk!" and the OS will reply "Sorry! No more disk space!"



Shmuel Csaba Otto Traian CC BY-SA 3.0 (), via Wikimedia Commons ()

OK, so now you know what an Operating System is. Linux is one member of a huge family of operating systems descended from something called Unix ().

Linux-based OSes serve the same essential purpose as Microsoft's Windows or Apple's OS X (also a Unix-derivative!), but it's a very different animal in some important ways. For one thing, its code is freely available to everyone. For another, there is no single, "official" form of the Linux operating system. Instead, many groups offer versions for different needs and purposes.

They may not be directly visible, but you probably use lots of computers running Linux, or at least the Linux kernel. That's because it runs on millions of the machines that make up our networks and infrastructure: Systems ranging from tiny embedded devices to enormous supercomputers used for scientific work, and nearly everything in-between.

For example, here're some common products/devices that run Linux:

- Android phones and tablets
- 40-80% of the public servers on the internet () (including this one!)
- Kindle and many other e-readers ()
- WiFi home gateways / routers ()
- Video game systems ()
- DVRs ()

There's a pretty good chance you already have a Linux machine or two in your house — or your pocket.

Do you need Linux for your projects?

Now that you know what Linux is, you might be thinking "Wow that's great, it would be really nice to run 'programs' on my Arduino instead of messing around with interrupts for when I want to drive Servos and NeoPixels at the same time ()".

Do you need to use Linux for your projects?

Maybe not, for a lot of things! A full modern operating system is overkill for many problems, and lots of simple, modular hardware tools are available these days. It's easier than it ever has been to build a gadget using Arduino+sensors+display and teach it what you need it to do.

On the other hand, little computers that run Linux well can be had for less than $50, and Linux is an incredibly powerful platform.

When you DONT want to use Linux

OK this isn't a hard and fast list of rules, but more stuff to keep in mind when you are deciding whether to use a Linux-based computer or go with something embedded/"bare metal"

- Power requirements are way higher for 'computers' - a microcontroller can use under a milliWatt, its tough to get a linux computer under 1 Watt. You may also have cooling management to deal with if your linux board is running 'hot'
- Speed-to-boot: if you don't have a bootloader, a microcontroller will start running its 'program' in milliseconds or less. Linux computers tend to require at least a minute, although you can sometimes tweak this down or it can take much more
- Disk Corruption: barring cosmic rays (), once a program is burned into the FLASH of an Arduino chip, it's there for life. You can turn it on 50 years from now and it'll be exactly the same. Linux computer disk drives can get corrupted by power fluctuations and can 'fail to boot' if data is not written correctly (there are ways to avoid this but its a bit of work and never completely foolproof)
- Size: Computers need a lot of stuff like FLASH, RAM, bus managers, multiple power supply regulators, they'll be bigger physically

- Price: All those parts cost $ too! You can DIY an Arduino for a few $, but its hard to get a linux computer for under $25. It's not much but if you need to make a lot of your project, it adds up!
- "Real Time" needs: If you are trying to perform a task with very specific timing requirements, like "precisely 0.5 milliseconds after this button is pressed, activate the flash bulb", a microcontroller will excel because it does not have to kernel running around from task to task. It will listen for the button, then wait for the exact amount of time. With linux, the kernel is doing all this stuff in the background, so many timing-specific stuff can be a challenge.
- Complexity: With so many interplaying parts, a lot more can go wrong...

When you WANT use Linux

A robust, modern OS makes it easy to speak network protocols (TCP/IP! SSL!), process large/complicated files like compressed photos/music/video, use low-cost off-the-shelf hardware that talks over USB, and grants access to countless different programming languages and other tools:

- A traditional Unix shell and the GNU core utilities ()
- Languages like: C/C++, Go, Python, Ruby, Perl, Scheme, PHP, and Java
- Frameworks like: Node.js (including libraries written especially for embedded development ()) and Flask ()
- Package managers and repositories like apt (), npm (), CPAN (), pip, Packagist (), and RubyGems (), offering fast access to tens of thousands of open source applications and libraries
- A full TCP/IP network stack and toolset
- Easy and familiar file storage to disk drives and SD cards
- Mature databases like PostgreSQL (), MySQL, and SQLite ()
- Web servers like nginx (), Apache (), and lighthttpd ()

Linux is a gateway to half a century's worth of accumulated knowledge, and a bridge from your work to the wider world.

It's also — and this is important — totally within your reach to learn. In the rest of this series, we'll provide a gentle introduction to basic computing with Linux, touch on the history of its key ideas, and explore how those ideas can be applied in creative work.

What about choosing a distribution?

OK now you have decided you are going to use Linux, you aren't done! Now you have to pick the flavor of Linux, also known as a Linux distribution.

We'll be using the Raspberry Pi for most examples throughout this series, and it has its own custom version of Debian GNU/Linux (), called Raspbian (a portmanteau () of Raspberry+Debian ()) which is a reliable operating system with tons of pre-packaged software available. I recommend you start with this distribution since it is popular, easy to get started with, and well maintained, but I'll discuss other options in a moment.

Wait! What's a distribution?

First, it's helpful to know what a "distro", short for "distribution" is: The Linux kernel bundled up and distributed with a whole bunch of other software to create a complete operating system that you can actually use.

This is kinda like how if you have a Windows or Mac computer, it comes with a basic text editor (SimpleText or WordPad), a game or two, a web browser (like Safari or Internet Explorer), and more.

...and so on and so forth. It's a lot of work, assembling an operating system.

A distribution represents a set of choices about how an OS should fit together and what things it should do well. It usually comes with a system for updating itself and installing new software, and how this problem is solved is one of the defining qualities of a distribution.

There are a lot of different distributions () out there. Some of them have been around for decades; others are only months old. It's a constantly changing landscape, and it can be easy to get overwhelmed by all the options.

Choosing a distribution is often just a question of deciding what your priorities are for a project. Some place a special emphasis on including only free software, being a friendly desktop system, or security. Others are meant to run on specific kinds of computers, like small low-power devices or warehouses full of servers. There's over 100 different Linux distributions () you can choose from (although not all of them may support the Pi or BBB out of the box)

Picking a distro for the Pi or BeagleBone

For little computers like the Raspberry Pi and the Beaglebone Black, distributions are available for tasks like running a media center (), building an arcade game (), testing the security of a school or company network (), developing hardware projects, etc. etc.

In general, we suggest sticking with the default distribution that comes with or is suggested for a computer. For Pi that's going to be Raspbian, for BBB that would be Debian, for a desktop or laptop computer, Ubuntu or Debian are popular.

We'll explore some of those in depth in later articles in this series. For now, let's...

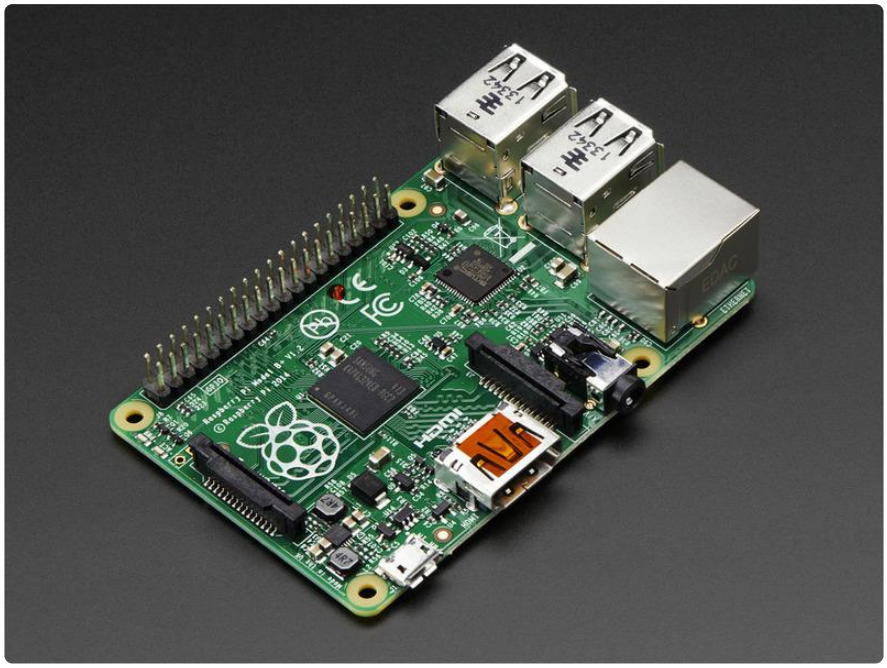
Get you a Linux Computer

If you're interested in giving Linux a try, there are a lot of options. I'll break down three of the easiest.

Buy a single-board computer

One of the easiest things you can do at this stage of history is buy a low-cost, standalone computer designed to run GNU/Linux.

Option 1 - Raspberry Pi



The Raspberry Pi is a simple, tiny device intended for educational environments and the learning of basic computer science topics. If you have a keyboard, mouse, and TV or monitor, you can plug directly into it.

We'll be using the Pi as a baseline system for articles in this introductory series because it's easy to come by, designed for teaching, and has a great community.

The Adafruit Learning System offers an ongoing series of introductory Raspberry Pi articles (), including detailed instructions on preparing the Pi for first boot () and first time configuration ().

Best of all it's low cost enough, you can get started for $100 - less if you already have some of this stuff around. We recommend at least:

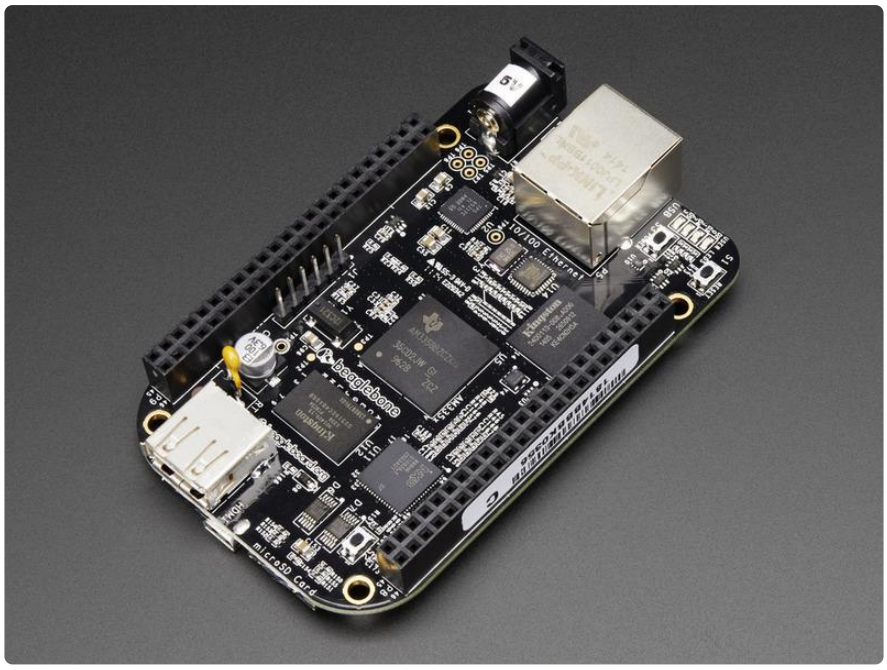
- Raspberry Pi Model B+ ()
- A case for for the Pi ()
- A good power supply (http://adafru.it/1995) for the Pi
- 4GB SD card pre-burned with the Raspbian OS (http://adafru.it/1121)
- WiFi USB Stick (http://adafru.it/1012) to get the Pi online
- USB Console Cable (http://adafru.it/954) so you can talk to the Pi from a desktop or laptop

If you want to get a pack with all this and some other handy parts, pick up:

Raspberry Pi Model B+ Starter Pack

http://adafru.it/2125

Option 2 - BeagleBone Black



The BeagleBone Black is, like the Raspberry Pi, a little computer on one board. It features a substantially faster processor, onboard storage, and a broader array of hardware interfaces than the Pi. It also lets you connect to a nice web-based interface over a USB cable out of the box, which means you can get started without a spare keyboard & monitor or messing around with network configuration.

On the other hand, it has fewer USB ports and no headphone jack, and offers less in the way of graphics capability. The supply has also been somewhat constrained, making it a harder gadget to come by, and fewer units in the wild means there's not as much written about using it, so there are not as many tutorials and projects. In that sense, the BBB is more "cutting edge".

The BBB is a beautiful little device, and you probably won't go wrong getting one. It might be most appropriate for cases where it's not going to be the only standalone computer in a room, or where you plan to do a lot of hardware hacking.

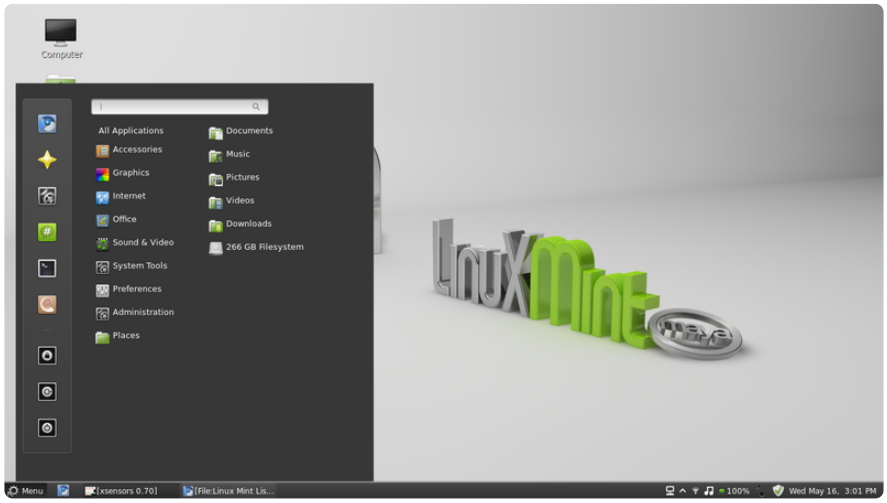
We've also got you covered on tutorials for the BeagleBone (), and the System Reference Manual () is a quality old-school manual, if you're into that kind of thing.

Beaglebone Black Starter Pack

http://adafru.it/703

If you're still not sure what to buy and want to get into the nitty-gritty, we've also got a detailed comparison of popular embedded Linux boards ().

Use your existing desktop or laptop



If you're reading this, there's a fair chance you already have a machine capable of running Linux well.

The traditional approach is to boot Linux directly on your machine. If you want to go this route, the easiest option is probably to install Ubuntu (), a widely-used close relative of Debian which supports a lot of recent hardware.

Another option here is to run a distribution on a virtual machine on top of your existing operating system. VirtualBox () is a widely-used, open source way to host other operating systems on x86 computers, including Mac and Windows systems. To make this easy, you can use Vagrant () to quickly spin up a new virtual machine - check out the first page of their Getting Started guide () and you should be Linuxing in no time.

Run Linux in the cloud

You don't have to use your own hardware to get a virtual machine — lots of services exist to host them for you.

Big companies you've heard of, like Amazon and Google, offer lots of cloud services. A lot of the internet runs on these, especially Amazon's offerings.

There's a lot of good material on using the cloud, but it can all be a bit overwhelming. Right now, I'll suggest a provider I've been using lately called DigitalOcean (), who offer an easy signup process, a simple web interface, and lease basic machines for about $5 a month.

Check out our guide to setting up a DigitalOcean droplet in about 10 minutes ().

Ready? Set...

We hope this tutorial helped you understand what we mean by operating system, kernel, Linux, and distribution! It's a lot to learn so keep in mind you may have to refer back to this tutorial in the future when we chat about kernel modules and package management.

Next up, we'll be exploring the command line () and demonstrating how to get real work done with the Linux toolset - so boot up your Linux system and sew that Tux patch onto your jacket! ()