

# A guide to UNIX and its users

October 9, 2003

Ben Klemens

This paper describes what those people who use UNIX are thinking. First, I hope that this will make peace between people who use different systems. Maybe it's just me, but too much of the world's GDP is wasted in arguments about operating systems.

Second, this serves as an introduction to using any computer or program that requires typing lots of commands. The people who wrote those programs think in a pretty consistent manner, and if you can understand what they were thinking, then the rest is just details. If I may brag, I'm sometimes tempted to list on my resume, 'Programming languages with which I am familiar: All of them', because in the end, only the details change from one to the next.

I primarily have click'n'draggers in mind, but the command typers will also benefit from having a framework to better explain him/herself, and with which to understand why their friends still use Windows even though it crashes every twenty minutes.

## 1 Applying Occham's Razor

The problem is in that simplicity is a good thing, but a computer system that can do anything useful has to be pretty complex. There are two ways to reconcile this, what we could describe as top-down and bottom-up. The top-down approach is that taken by the Windows people: put together something that looks simple to the user up at the top level, and then work out how you go about implementing that in terms of hardware instructions and all that crap.

The bottom-up approach (taken by the UNIX<sup>1</sup> people) is to write the hardware instructions and everything in as simple a manner as possible, and let the system evolve from there to

---

<sup>1</sup>UNIX is a trademark of AT&T, since it was developed at Bell Labs. There are a dozen systems which work just like UNIX, like Linux, AIX, SunOS, and anything else that's labelled as 'POSIX standard compliant'. Without apology, I'm going to call them all UNIX.

do more as necessary. The result looks a lot like a math textbook.

## 1.1 math textbooks

Euclid's Elements is the most famous of the Greek treatises on geometry, but it turns out that it is by no means the only one. The many entrants into this popular Greek genre are based on the same conceit: to begin with as few axioms as possible and derive as much as possible. ¿Is there any *a priori* reason to presume that such a setup would be more robust or True than another setup which made many reasonable assumptions all along the way but required fewer steps of logic to reach a conclusion? I'm no philosopher, but not really. The big advantage that the 'few axioms and many steps of logic' approach has is the æsthetic characteristic which we refer to as 'neat'. Neat in the sense that a machine with few moving parts is neat, and neat in the sense that a clean apartment without too many knick-knacks is neat.

Of course, the cost to this neatness is 13 books' worth of logic. For many applications, it's easier to just assume some intermediate steps and go from there. A lot of people like their knick-knacks, and Euclid certainly doesn't have any proof that they shouldn't.

## 1.2 C

So, ¿what are the basic principles on which one could write a computer system? I won't go into detail about hardware interfaces, the /dev devices, blahdetechblah, but the main thing to note here is the extensive use of the C programming language. A C program consists of a handful of symbols, an assortment of variable names which are basically pneumonics, and a vocabulary of about twenty keywords. Those words are the axioms on which the system is based, and although Euclid got by with five, twenty ain't bad.

So, ¿what kind of programs do you write? Keeping with the idea of building from simple parts, the bottom-up designer writes programs which are as simple and as possible, and which can be strung together to form bigger, more complex programs. This is where some people start to think the bottom-up approach is a pain. At the command prompt of my computer, I can type 1,500 different commands. I probably know what maybe ten percent of them do. Between now and when I die, I'm not going to do much better.

Few people have such a level of acceptance toward the stuff that fills their life. Although it's sometimes amusing to find a new dial in the back of the fridge which controls the lettuce crisper, folks generally don't take kindly to surprises. They want a manual, and the reassurance that the manual will tell them everything there is. If you go to B Dalton's or whatever, there's that shelf of thick manuals, which, *en masse*, hold 8% of the earth's

biomass,<sup>2</sup> filling this need.

So when I was a kid there was a ‘forest’ behind our house, and we never really found the end of it. Every time my brother Guy and I went there, there was somewhere else to go. When I went to visit my young-enough-to-be-my-kids half-brother and half-sister outside London, there was an undeveloped part behind their house, and over the course of half an hour, we went down every trail, and I established that the wood was just the center of the block and it was completely surrounded by houses. I don’t think the kids minded, but was I ever disappointed.

There’s not much left to explore as an adult in the modern world. It often happens that a new neighborhood or niche of Internet quickly becomes more of the same, as you go from ‘Koreatown is so charming’ to ‘Oh look. Another Korean restaurant’. Advertising which says ‘Discover our product’ is just depressing.

¿So why is having 1,500 different commands to type different? After all, every one of them either writes crap to your hard drive or to your screen. The first distinction is that every one of those commands is a verb. There are 1,500 things I can *do* at the command prompt. There’s a sense not of disempowerment in having a computer which knows more than I do, but of empowerment that there’s so much that *I* can do.

The other point about having 1,500 very small programs is that they afford a huge amount of control. The classic example of this is `ls`, the ‘list directory’ command. In a MSFT Windows sort of setup, you can set the explorer to list your directories by icon or list, and change a handful of other features. The `ls` command has over thirty different options. So forget just listing files in a directory willy-nilly. I have complete, exact control of what appears on the screen.<sup>3</sup>

The comfort one can draw from this is obvious. As discussed, you’ve got a system far more complex than one human brain can conceive of, and on the other hand, you can get it to do exactly what you want it to. I don’t think that there’s much that’s comparable.

### 1.3 snobbery

The accusation has often been made that the UNIX geek is a snob, who gets off on the idea of knowing an assortment of obscure commands which the poor click’n’dragger will always (choose to) remain ignorant of. This is often an artifact of the power and control which I

---

<sup>2</sup>Cite: my friend Jon.

<sup>3</sup>As a technical aside, ¿How does somebody manage thirty options? The answer is of course the manual, which is online via the `man` command. Usually, you look up `man ls` to get all the switches to choose from, pick your favorites, and then write a one-line script in the way of `alias ls = ls -blah` so that every time you type `ls`, options `b`, `l`, `a`, and `h` get selected. Ideally, you never look at the man page again and entirely forget what those options do.

discussed here. If you value such things, it's hard to conceive of how people deal with those frustrating systems that aren't as powerful and configurable. The argument often begins with the geek pointing out, 'I can do this and that and those', and then the non-geek replies, '¿So what?'

The bottom-up system certainly looks more complicated, since the simple part is way in the internals of the stuff the user doesn't really deal with directly. This means that to talk about things that seem like simple ideas requires a lot of little commands and techniques that the top-down user never really thinks about. But talking like this is not an attempt to look smart, it's just how things are done.

OK, for some people, it's an attempt to look smart.

The other root of snobbery is the '¿why don't you get it?' question. One good indicator of why a bottom-up system appeals to a person is that it's easy to deal with. The math textbook metaphor becomes more than a metaphor here: if you understood your geometry classes well, then you've already got the mindset that the folks who wrote UNIX had. It's easy to work up the learning curve of knowing the framework and how all those little commands fit together—and it's hard to work out how other people can't work it out. But the question of how one teaches a mindset is one which math teachers have been grappling with for millenia and have not yet solved. This engenders snobbery more than any jargon of commands ever could, because it's so easy to say, 'I get it and you don't, so I'm a better person.' This is only true to the extent that doing well in high school geometry indicates that one is a good person.

## 2 text

Here we are, literate people. Why, right now, you're reading words. Although an incredibly complicated process which Modern Science has not yet fully got a handle on, most of us consider it to be trivial and, within the context here, a low-level sort of operation. And so, text lies toward the base of the bottom-up systems.

The method of recording text is ASCII, the American Standard Code for Information Interchange. ASCII is a miracle of consensus: every computer on the planet (OK, except for some old IBM mainframes) can display text in ASCII format. If the UN worked this well, we'd never have war again.

The idea is pretty simple: people read letters, computers only store numbers, so we need a standard for converting between the two. ASCII provides this: A=65, B=66, et cetera.<sup>4</sup>

---

<sup>4</sup>1 through 64 are technical codes, punctuation, and numbers. Lowercase gets its own sequence: a=97, b=98, et cetera.

Oh, ¿what could be simpler?

So a UNIX system is filled with commands for dealing with text. There's `more` to display a page of text, so named because at the bottom of the page it says something like 'More—3%' to tell you that you're only 3% through the text and should page down. There's the sequel, `less`, which does a few things better than `more`, and is so named because its authors were smartasses. There's `grep` which will let you search for stuff.<sup>5</sup> If you want to actually edit your text, there's `vi`, `ed`, `emacs`, `pico`, `joe`, `elvis`, et cetera, et cetera.

This is good enough for 90% of what we humans want to do. Your address book is just words and numbers (and is easy to search with `grep`); your letters are just words; the web pages you spend all day at work looking at are just words.

'But they're not,' you retort, 'They include formatting and cute pictures.' I'll concede the pictures (you pervert), but the formatting is easy to handle: we can just insert text to specify what your web browser should do. Your browser should have a 'view source' option in there somewhere. It'll show that your web pages are written in plain text, something like this:

```
<h1>Headline!</h1>
```

```
It is of the <i>utmost</i> that you read <u>A guide to UNIX and its users</u>.
```

'Headline!' will be the header, and the sentence will look like this: It is of the *utmost* that you read A guide to UNIX and its users. All you need is a web browser to understand that `<i>` means italicize, and you're done: you've specified all the wonderful formatting you could want using plain text. This paper is written in  $\text{\TeX}$ , which uses a similar protocol. Here's what I wrote:

```
It is of the {\it utmost} that you read \underline{A guide to UNIX and its users}.
```

¿Why is this useful?, you ask. Well, it gets back to the bottom-up versus top-down paradigm. The bottom-up paradigm loves text because it is at the lowest level—it's how we think. It is robust: if a few letters somehow get mangled, we humans can easily rewrite them based on context. It is easy to edit because all of these different formats, HTML for the browser,  $\text{\TeX}$  for the written documents,  $\text{C}^{++}$  for the programs which interpret text, can all be handled using the same text editing programs which handle your address book, your love letters, and your business emails.

---

<sup>5</sup>So named because it is an acronym for general regular expression parser. Regular expressions are patterns which you want to match. E.g., `stuff[- ]*` will look for any word beginning with 'stuff', like 'stuffed', 'stuffing', et cetera. Consistent with the paradigm, there's a lot to learn before you're good with regexps, but once you have them, the world is yours for the taking. When people ask me what they can do to get better with UNIX (really, people ask me these things), I tell them to learn regular expressions.

The cost is that you have to learn all those formatting codes. Personally, I think this is easy: 90% of what you'll do in  $\text{\TeX}$  is italicize, underline, and put up section headers, and those four or five commands won't take up much of your brain.

The top-down person disagrees, and wants the highest-level, what the user sees, to be as simple as possible. The aesthetic advantages hit the eye immediately: you spend your time looking at a picture of a page, which is much more pleasing to the eye than raw text will ever be. So there's a button you press to italicize, and it doesn't matter how the computer represents it internally, since the user never sees it. Your next homework assignment is to open a MSFT Word document using the Notepad. You'll see gibberish: mostly blank squares signifying unprintable characters, with perhaps a few pieces of legible text here and there. This has a few technical advantages: you can shrink the size of the file using compression tricks; you can optimize how this is read into the computer's memory; you can perhaps more easily take notes on things like user settings in a way that would be cumbersome (but not impossible) using plain text.

It also has problems, the counterpart to the bottom-up system's advantages. First and foremost, it's not human-readable unless you have the program the file is intended for. We've all faced this problem: you get handed a file written for Word 97 but only have a copy of Word 95—or still worse, WordPerfect. In such a situation, there's nothing you can do but run out and purchase a copy of Word 97. Until you do this, 100% of the information in the file is lost to you. Back in the days of floppy disks, it happened to me all the time that a few bits in a Word file would get corrupted, and then Word refused to read the whole file. Meanwhile, barring nuclear apocalypse, we'll probably be able to read an ASCII text file a decade from now with no problems.

## 2.1 the text editor

So in a bottom-up system all sorts of files for all sorts of purposes are all text. All this means that UNIX users spend 90% of their day either at the command prompt or in their text editors. The command prompt has been discussed extensively above. As for the average UNIX text editor, there's nothing like it in the top-down world. Since people spend all day in them, they're very, very well optimized. For example, the `vi` editor is a masterpiece of ergonomics because your hands never leave the home position. That half-second it takes for you to move your hand down to the arrow keys when typing in a word processor adds up to a minute or two over the course of a day—a minute that you could be spending staring into space or going to the bathroom again.

I won't go into detail about how this masterwork functions, but needless to say, there's a learning curve—which is again anathemic to the top-down approach. But once the bottom-up user has worked up the curve, they have an incredibly efficient system: the text editor does nothing but edit text, so all of its commands are focused on that, and everything you'd

want to do can be done. ¿Want to change the ‘b’ six lines up to a ‘B’? That’s five keystrokes,<sup>6</sup> while in your average top-down editor, it’d be six bangs on the up-arrow just to get to that line.

Some people feel resignation at the thought of having to fill their brain with trivia about keystrokes. Computers are taking over our lives, and many of us aren’t happy with that. If we fill our grey matter with more commands, formatting codes, and keystrokes, then the computers have won and we’re working for them instead of the other way around.

On the other hand, the forward progress of those little PCs is inevitable, and soon enough everybody will have to deal with one at work and another at home. An alternate approach is to just accept this fact, and find ways to deal with these machines in as efficient a manner as possible, even though that means spending all day at first learning the dumb tricks. Some go further and embrace the takeover, and are enthused by the empowerment that facility with these machines can bring.

### 3 Conclusion

Those of you who have been magically sold on the bottom-up paradigm now know where to begin. There are a few basic commands you need to know to get around the system; you need to pick a text editor and get to know it very well; you’ll want to know a program like T<sub>E</sub>X so you can write your papers and love letters with neat formatting; and you will probably find benefit from learning one of the bazillion programming languages which come standard with most versions of UNIX.

For those of you who haven’t been magically sold, I hope you have a better understanding of the bottom-up user: they just want to work as efficiently as possible, and want complete control of their work, and are willing to pay the initial cost of learning dumb commands to get there. The æsthetics of the system lie not in the screen that the user faces, but in the conceptual structure upon which the system is based.

This paper is currently available at [http://avocado.caltech.edu/klemens/unix\\_guide.pdf](http://avocado.caltech.edu/klemens/unix_guide.pdf) .

---

<sup>6</sup>kFb~