Things Every Hacker Once Knew

by Eric S. Raymond <esr@thyrsus.com>

Table of Contents

One fine day in January 2017 I was reminded of something I had half-noticed a few times over the previous decade. That is, younger hackers don't know the bit structure of ASCII and the meaning of the odder control characters in it.

This is knowledge every fledgling hacker used to absorb through their pores. It's nobody's fault this changed; the obsolescence of hardware terminals and the near-obsolescence of the RS-232 protocol is what did it. Tools generate culture; sometimes, when a tool becomes obsolete, a bit of cultural commonality quietly evaporates. It can be difficult to notice that this has happened.

This document began as a collection of facts about ASCII and related technologies, notably hardware serial terminals and RS-232 and modems. This is lore that was at one time near-universal and is no longer. It's not likely to be directly useful today - until you trip over some piece of still-functioning technology where it's relevant (like a GPS puck), or it makes sense of some old-fart war story. Even so, it's good to know anyway, for cultural-literacy reasons.

One thing this collection has that tends to be indefinite in the minds of older hackers is *calendar dates*. Those of us who lived through all this tend to have remembered order and dependencies but not exact timing; here, I did the research to pin a lot of that down. I've noticed that people have a tendency to retrospectively back-date the technologies that interest them, so even if you did live through the era it describes you might get a few surprises from reading this.

There are lots of references to Unix in here because I am mainly attempting to educate younger open-source hackers working on Unix-derived systems such as Linux and the BSDs. If those terms mean nothing to you, the rest of this document probably won't either.

Hardware context

Nowadays, when two computers talk to each other, it's usually via TCP/IP over some physical layer you seldom need to care much about. And a "terminal" is actually a "terminal emulator", a piece of software that manages part of a bit-mapped display and itself speaks TCP/IP.

Before ubiquitous TCP/IP and bit-mapped displays things were very different. For most hackers that transition took place within a few years of 1992 - perhaps somewhat earlier if you had access to then-expensive workstation hardware.

Before then there were video display terminals - VDTs for short. In the mid-1970s these had displaced an earlier generation of printing terminals derived from **really** old technology called a "teletype", which had evolved around 1900 from Victorian telegraph networks. The very earliest versions of Unix in the late 1960s were written for these printing terminals, in particular for the Teletype Model 33 (aka ASR-33); the "tty" that shows up in Unix device names was a then-common abbreviation for "teletype".

(This is not the only Unix device name that is a fossil from a bygone age. There's also /dev/lp for the system default printer; every hacker once knew that the "lp" stood for "line printer", a type of line-at-a-time electromechanical printer associated with mainframe computers of roughly the same vintage as the ASR-33.)

One legacy of those printing terminals is the "Carriage return" character (Ctrl-r, or "\\r" expressed as a C string escape), so called because it moved the print head to the left hand end of the line. Unix dispensed with it, using a bare line feed (Ctrl-N or "\\n") as an end-of-line marker. Many other operating systems still use \r\n. The reason it's \r\n rather than the \n\r that the \r actually took more than one normal chaacter-transmission time to execute on some of the earliest hardware; you had to do it first so it could finish as the \n was being performed.

It's half-forgotten now that these VDTs were deployed in fleets attached to large central machines. Today a single personal computer has multiple processors, but back then all the VDTs and their users on a machine divided up a single processor; this was called "timesharing". Modern Unixes still have this capability, if anyone still cared to use it; the typical setup of having many "virtual terminals", accessible by pressing Ctrl-Alt-Fsomething, uses the very mechanism that originally provided access to many terminals for timesharing.

In those pre-Internet days computers didn't talk to each other much, and the way teletypes and terminals talked to computers was a hardware protocol called "RS-232" (or, if you're being pedantic, "EIA RS-232C"). [1] Before USB, when people spoke of a "serial" link, they meant RS-232, and sometimes referred to the equipment that spoke it as "serial terminals".

RS-232 had a very long service life; it was developed in the early 1960s, not originally for computer use but as a way for teletypewriters to talk to modems. Though it has passed out of general use and is no longer common knowledge, it's not quite dead even today.

I've been simplifying a bit here. There were other things besides RS-232 and serial terminals going on, notably on IBM mainframes. But they've left many fewer traces in current technology and its folklore. This is because the lineage of modern Unix passes back through a now-forgotten hardware category called a "minicomputer", especially minicomputers made by the Digital Equipment Corporation. ASCII, RS-232 and serial terminals were part of the technology cluster around minicomputers - as was, for that matter, Unix itself.

Minicomputers were wiped out by workstations and workstations by descendants of the IBM PC, but many hackers old enough to remember the minicomputer era (mid-1960s to mid-1980s) tend still to get a bit misty-eyed about the DEC hardware they cut their teeth on.

Often, however, nostalgia obscures how very underpowered those machines were. For example: a DEC VAX 11-780 minicomputer in the mid-1980s, used for timesharing and often supporting a dozen simultaneous users, had less than 1/1000 the processing power and less than 1/5000 times as much storage available as a low-end smartphone does in 2017.

In fact, until well into the 1980s microcomputers ran slowly enough (and had poor enough RF shielding) that **this** was common knowledge: you could put an AM radio next to one and get a clue when it was doing something unusual, because either fundamentals or major subharmonics of the clock frequencies were in the 20Hz to 20KHz range of human audibility. Nothing has run **that** slowly since the turn of the 21st century.

The strange afterlife of the Hayes smartmodem

About those modems: the word is a portmanteau for "modulator/demodulator". Modems allowed digital signals to pass over copper phone wires - ridiculously slowly by today's standards, but that's how we did our primitive wide-area networking in pre-Internet times. It was **not** generally known back then that modems had first been invented in the late 1950s for use in military communications, notably the SAGE air-defense network; we just took them for granted.

Today modems that speak over copper or optical fiber are embedded invisibly in the Internet access point in your basement; other varieties perform over-the-air signal handling for smartphones and tablets. A variety every hacker used to know about (and most of us owned) was the "outboard" modem, a separate box wired to your computer and your telephone line.

Inboard modems (expansion cards for your computer) were also known (and became widespread on consumer-grade computers towards the end of the modem era), but hackers avoided them because being located inside the case made them vulnerable to RF

noise, and the blinkenlights on an outboard were useful for diagnosing problems. Also, most hackers learned to interpret (at least to some extent) modem song - the outboards made while attempting to establish a connection. The happy song of a successful connect was identifiably different from various sad songs of synchronization failure.

One relic of modem days is the name of the Unix SIGHUP signal, indicating that the controlling terminal of the user's process has disconnected. HUP stands for "HangUP" and this originally indicated a serial line drop (specifically, loss of Data Carrier Detect) as produced by a modem hangup.

These old-fashioned modems were, by today's standards, unbelievably slow. Modem speeds increased from 110 bits per second back at the beginning of interactive computing to 56 kilobits per second just before the technology was effectively wiped out by wide-area Internet around the end of the 1990s, which brought in speeds of a megabit per second and more (20 times faster). For the longest stable period of modem technology after 1970, about 1984 to 1991, typical speed was 9600bps. This has left some traces; it's why surviving serial-protocol equipment tends to default to a speed of 9600bps.

There was a line of modems called "Hayes Smartmodems" that could be told to dial a number, or set parameters such as line speed, with command codes sent to the modem over its serial link from the machine. Every hacker used to know the "AT" prefix used for commands and that, for example, ATDT followed by a phone number would dial the number. Other modem manufacturers copied the Hayes command set and variants of it became near universal after 1981.

What was **not** commonly known then is that the "AT" prefix had a helpful special property. That bit sequence (1+0 1000 0010 1+0 0010 1010 1+, where the plus suffix indicates one or more repetitions of the preceding bit) has a shape that makes it as easy as possible for a receiver to recognize it even if the receiver doesn't know the transmit-line speed; this, in turn, makes it possible to automatically synchronize to that speed [2].

That property is still useful, and thus in 2017 the AT convention has survived in some interesting places. AT commands have been found to perform control functions on 3G and 4G cellular modems used in smartphones. On one widely deployed variety, "AT+QLINUXCMD=" is a prefix that passes commands to an instance of Linux running in firmware on the chip itself (separately from whatever OS might be running visibly on the phone).

Preserving core values

From about 1955 to 1975 - before semiconductor memory - the dominant technology in computer memory used tiny magnetic doughnuts strung on copper wires. The doughnuts were known as "ferrite cores" and main memory thus known as "core memory" or "core".

Unix terminology was formed in the early 1970s, and compounds like "in core" and "core dump" survived into the semiconductor era. Until as late as around 1990 it could still be assumed that every hacker knew from where these terms derived; even microcomputer hackers for which memory had always been semiconductor RAM tended to pick up this folklore rapidly on contact with Unix.

After 2000, however, as multiprocessor systems became increasingly common even on desktops, "core" increasingly took on a conflicting meaning as shorthand for "processor core". In 2017 "core" can still mean either thing, but the reason for the older usage is no longer generally understood and idioms like "in core" may be fading.

36-bit machines and the persistence of octal

There's a power-of-two size hierarchy in memory units that we now think of as normal - 8 bit bytes, 16 or 32 or 64-bit words. But this did not become effectively universal until after 1983. There was an earlier tradition of designing computer architectures with 36-bit words.

There was a time when 36-bit machines loomed large in hacker folklore and some of the basics about them were ubiquitous common knowledge, though cultural memory of this era began to fade in the early 1990s. Two of the best-known 36-bitters were the DEC PDP-10 and the Symbolics 3600 Lisp machine. The cancellation of the PDP-10 in '83 proved to be the death knell for this class of machine, though the 3600 fought a rear-guard action for a decade afterwards.

Hexadecimal is a natural way to represent raw memory contents on machines with the power-of-two size hierarchy. But octal (base-8) representations of machine words were common on 36-bit machines, related to the fact that a 36-bit word naturally divides into 12 3-bit fields naturally represented as octal. In fact, back then we generally assumed you could tell which of the 32- or 36-bit phyla a machine belonged in by whether you could see digits greater than 7 in a memory dump.

Here are a few things every hacker used to know that related to these machines:

- 36 bits was long enough to represent positive and negative integers to an accuracy of ten decimal digits, as was expected on mechanical calculators of the era. Standardization on 32 bits was unsuccessfully resisted by numerical analysts and people in scientific computing, who really missed that last 4 bits of accuracy.
- A "character" might be 9 bits on these machines, with 4 packed to a word. Consequently, keyboards designed for them might have both a meta key to assert bit 8 and a now-extinct extra modifier key (usually but not always called "Super") that asserted bit 9. Sometimes this selected a tier of non-ASCII characters including Greek letters and mathematical symbols.
- Alternatively, 6-bit characters might be packed 6 to a word. There were many different 6-bit character encodings; not only did they differ across a single manufacturer's machines, but some individual machines used multiple incompatible encodings. This is why older non-Unix minicomputers like the PDP-10 had a six-character limit on filenames - this allowed an entire filename to be packed in a single 36-bit word. If you ever see the following acronyms it is a clue that you may have wandered into this swamp: SIXBIT, FIELDATA, RADIX-50, BCDIC.

It used also to be generally known that 36-bit architectures explained some unfortunate features of the C language. The original Unix machine, the PDP-7, featured 18-bit words corresponding to half-words on larger 36-bit computers. These were more naturally represented as six octal (3-bit) digits.

The immediate ancestor of C was an interpreted language written on the PDP-7 and named B. In it, a numeric literal beginning with 0 was interpreted as octal.

The PDP-7's successor, and the first workhorse Unix machine was the PDP-11 (first shipped in 1970). It had 16-bit words - but, due to some unusual peculiarities of the instruction set, octal made more sense for its machine code as well. C, first implemented on the PDP-11, thus inherited the B octal syntax. And extended it: when an in-string backslash has a following digit, that was expected to lead an octal literal.

The Interdata 32, VAX, and other later Unix platforms didn't have those peculiarities; their opcodes expressed more naturally in hex. But C was never adjusted to prefer hex, and the surprising interpretation of leading 0 wasn't removed.

Because many later languages (Java, Python, etc) copied C's low-level lexical rules for compatibility reasons, the relatively useless and sometimes dangerous octal syntax besets computing platforms for which three-bit opcode fields are wildly inappropriate, and may never be entirely eradicated [3].

The PDP-11 was so successful that architectures strongly influenced by it (notably, including Intel [4] and ARM microprocessors) eventually took over the world, killing off 36-bit machines.

The x86 instruction set actually kept the property that though descriptions of its opcodes commonly use hex, large parts of the instructiion set are best understood as three-bit fields and thus best expressed in octal. This is perhaps clearest in the encoding of the mov instruction.

RS232 and its discontents

A TCP/IP link generally behaves like a clean stream of 8-bit bytes (formally, octets). You get your data as fast as the network can run, and error detection/correction is done somewhere down below the layer you can see.

RS-232 was not like that. Two devices speaking it had to agree on a common line speed - also on how the byte framing works (the latter is why you'll see references to "stop bits" in related documentation). Finally, error detection and correction was done in-stream, sort of. RS232 devices almost always spoke ASCII, and used the fact that ASCII only filled 7 bits. The top bit might be, but was not always, used as a parity bit for error detection. If not used, the top bit could carry data.

You had to set your equipment at both ends for a specific combination of all of these. After about 1984 anything other than "8N1" - eight bits, no parity, one stop bit - became increasingly rare. Before that, all kinds of weird combinations were in use. Even parity ("E") was more common than odd ("O") and 1 stop bit more common than 2 [5], but you could see anything come down a wire. And if you weren't properly set up for it, all you got was "baud barf" - random 8-bit garbage rather than the character data you were expecting.

This, in particular, is one reason the API for the POSIX/Unix terminal interface, termios(3), has a lot of complicated options with no obvious modern-day function. It had to be able to manipulate all these settings, and more.

Another consequence was that passing binary data over an RS-232 link wouldn't work if parity was enabled - the high bits would get clobbered. Other now-forgotten wide-area network protocols reacted even worse, treating in-band characters with 0x80 on as control codes with results ranging from amusing to dire. We had a term, "8-bit clean", for networks and software that didn't clobber the 0x80 bit. And we **needed** that term...

The fragility of the 0x80 bit back in those old days is the now largely forgotten reason that the MIME encoding for email was invented (and within it the well-known MIME64 encoding). Even the version of SMTP current as I write (RFC 5321) is still essentially a 7-bit protocol, though modern end points can now optionally negotiate passing 8-bit data.

But before MIME there was uuencode/uudecode, a pair of filters for rendering 8-bit data in 7 bits that is still occasionally used today in Unixland. In this century uuencoding has largely been replaced by MIME64, but there are places you can still trip over uuencoded binary archives.

Even the RS-232 physical connector varied. Standard RS-232 as defined in 1962 used a roughly D-shaped shell with 25 physical pins (DB-25), way more than the physical protocol actually required (you can support a minimal version with just three wires, and this was actually common). Twenty years later, after the IBM PC-AT introduced it in 1984, most manufacturers switched to using a smaller DB-9 connector (which is technically a DE-9 but almost nobody ever called it that). If you look at a PC with a serial port it is most likely to be a DB-9; confusingly, DB-25 came to be used for printer parallel ports (which originally had a very different connector) before those too were obsolesced by USB and Ethernet.

Anybody who worked with this stuff had to keep around a bunch of specialized hardware - gender changers, DB-25-to-DB-9 adapters (and the reverse), breakout boxes, null modems, and other gear I won't describe in detail because it's left almost no traces in today's tech. Hackers of a certain age still tend to have these things cluttering their toolboxes or gathering dust in a closet somewhere.

The main reason to still care about any of this (other than understanding greybeard war stories) is that some kinds of sensor and control equipment and IoT devices still speak RS-232, increasingly often wrapped inside a USB emulation. The most common devices that do the latter are probably GPS sensors designed to talk to computers (as opposed to handheld GPSes or car-navigation systems).

Because of devices like GPSes, you may still occasionally need to know what an RS-232 "handshake line" is. These were originally used to communicate with modems; a terminal, for example, could change the state of the DTR (Data Terminal Ready) line to indicate that it was ready to receive, initiate, or continue a modem session.

Later, handshake lines were used for other equipment-specific kinds of out-of-band signals. The most commonly re-used lines were DCD (data carrier detect) and RI (Ring Indicator).

Three-wire versions of RS-232 omitted these handshake lines entirely. A chronic source of frustration was equipment at one end of your link that failed to supply an out-of-band signal that the equipment at the other end needed. The modern version of this is GPSes that fail to supply their 1PPS (a high-precision top-of-second pulse) over one of the handshake lines.

Even when your hardware generated and received handshake signals, you couldn't necessarily trust cables to pass them. Cheap cables often failed to actually connect all 25 (or 9) leads end-to-end. Then there were "crossover" or "null modem" cables, which for reasons too painful to go into here crosswired the transmit and receive lines. Of course none of these variants were reliably labeled, so debugging an RS232 link with a random cable pulled out of a box often involved a lot of profanity.

Another significant problem was that an RS-232 device not actually sending data was undetectable without analog-level monitoring equipment. You couldn't tell a working but silent device from one that had come unplugged or suffered a connection fault in its wiring. This caused no end of complications when troubleshooting and is a major reason USB was able to displace RS-232 after 1994.

A trap for the unwary that opened up after about the year 2000 is that peripheral connectors labeled RS232 could have one of two different sets of voltage levels. If they're pins or sockets in a DB9 or DB25 shell, the voltage swing between 1 and 0 bits can be as much as 50 volts, and is usually about 26. Bare connectors on a circuit board, or chip pins, increasingly came to use what's called "TTL serial" - same signalling with a swing of 3.3 or (less often) 5 volts. You can't wire standard RS232 to TTL serial directly; the link needs a device called a "level shifter". If you connect without one, components on the TTL side will get fried.

RS-232 passed out of common knowledge in the mid- to late 1990s, but didn't finally disappear from general-purpose computers until around 2010. Standard RS-232 is still widely used not just in the niche applications previously mentioned, but also in point-of-sale systems diagnostic consoles on commercial-grade routers, and debugging consoles on embedded systems. The TTL serial variant is often used in the latter context, especially on maker devices.

WAN time gone: The forgotten pre-Internets

Today, the TCP/IP Internet is very nearly the only WAN (Wide-Area-Network) left standing. It was not always so. From the late '70s to the mid-1990s - but especially between 1981 and 1991 - there were a profusion of WANs of widely varying capability. You are most likely to trip over references to these in email archives from that time; one characteristic of it is that people sometimes advertised multiple different network addresses in their signatures.

Every hacker over a certain age remembers either UUCP or the BBS scene. Many participated in both. In those days access to the "real" net (ARPANET, which became Internet) was difficult if you weren't affiliated with one of a select group of federal agencies, military contractors, or university research labs. So we made do with what we had, which was modems and the telephone network.

UUCP stands for Unix to Unix Copy Program. Between its escape from Bell Labs in 1979 and the mass-market Internet explosion of the mid-1990s, it provided slow but very low-cost networking among Unix sites using modems and the phone network.

UUCP was a store-and-forward system originally intended for propagating software updates, but its major users rapidly became email and a thing called USENET (launched 1981) that was the ur-ancestor of Stack Overflow and other modern web fora. It supported topic groups for messages which, propagated from their point of origin through UUCP, would eventually flood to the whole network.

In part, UUCP and USENET were a hack around the two-tier rate structure that then existed for phone calls, with "local" being flat-rate monthly and "long-distance" being expensively metered by the minute. UUCP traffic could be relayed across long distances by local hops.

A direct descendant of USENET still exists, as Google Groups [6], but was much more central to the hacker culture before cheap Internet. Open source as we now know it germinated in USENET groups dedicated to sharing source code. Several conventions still in

use today, like having project metadata files named README and NEWS and INSTALL, became established there in the early 1980s - though at least README was older, having been seen in the wild back on the PDP-10.

Two key dates in USENET history were universally known. One was the Great Renaming in 1987, when the name hierarchy of USENET topic groups was reorganized. The other was the "September that never ended" in 1993, when the AOL commercial timesharing services gave its users access to USENET. The resulting vast flood of newbies proved difficult to acculturate.

UUCP explains a quirk you may run across in old mailing-list archives: the bang-path address. UUCP links were point-to-point and you had to actually specify the route of your mail through the UUCP network; this led to people publishing addresses of the form "...!bigsite!foovax!barbox!user", presuming that people who wanted to reach them would know how to reach bigsite. As Internet access became more common in the early 1990s, addresses of the form user@hostname displaced bang paths. During this transition period there were some odd hybrid mail addresses that used a "%" to weld bang-path routing to Internet routing.

UUCP was notoriously difficult to configure, enough so that people who knew how often put that skill on their CVs in the justified expectation that it could land them a job.

Meanwhile, in the microcomputer world, a different kind of store-and-forward evolved - the BBS (Bulletin-Board System). This was software running on a computer (after 1991 usually an MS-DOS machine) with one (or, rarely, more) attached modems that could accept incoming phone calls. Users (typically, just one user at a time!) would access the BBS using a their own modem and a terminal program; the BBS software would allow them to leave messages for each other, upload and download files, and sometimes play games.

The first BBS, patterned after the community notice-board in a supermarket, was fielded in Chicago in 1978. Over the next eighteen years over a hundred thousand BBSes flashed in and out of existence, typically run out of the sysop's bedroom or garage with a spare computer.

From 1984 the BBS culture evolved a primitive form of internetworking called "FidoNet" that supported cross-site email and a forum system broadly resembling USENET. There were also a few ports of UUCP to DOS personal computers, but none gained any real traction. Thus the UUCP and BBS cultures remained separate until both were ploughed under by the Internet.

During a very brief period after 1990, just before mass-market Internet, software with BBS-like capabilities but supporting multiple simultaneous modem users (and often offering USENET access) got written for low-cost Unix systems. The end-stage BBSes, when they survived, moved to the Web and dropped modem access. The history of cellar.org chronicles this period.

A handful of BBSes are still run by nostalgicists, and some artifacts from the culture are still preserved. But, like the UUCP network, the BBS culture as a whole collapsed when inexpensive Internet became widely available.

Almost the only cultural memory of BBSes left is around a family of file-transfer protocols - XMODEM, YMODEM, and ZMODEM - developed shortly before BBSes and primarily used on them. For hackers of that day who did not cut their teeth on minicomputers with native TCP/IP, these were a first introduction to concepts like packetization, error detection, and retransmission. To this day (2018), hardware from at least one commercial router vendor (Cisco) accepts software patches by XMODEM upload through a serial port.

Also roughly contemporaneous with USENET and the BBS culture, and also destroyed or absorbed by cheap Internet, were some commercial timesharing services supporting dialup access by modem, of which the best known were AOL (America Online) CompuServe, and GEnie; others included The Source and Prodigy. These provided BBS-like facilities. Every hacker knew of these, though few used them. They have left no traces at all in today's hacker culture.

One last tier of pre-Internets, operating from about 1981 to about 1991 with isolated survivals into the 2000s, was various academic wide-area networks using leased-line telephone links: CSNET, BITNET, EARN, VIDYANET, and others. These generally supported email and file-transfer services that would be recognizable to Internet users, though with different addressing schemes and some odd quirks (such as not being 8-bit clean). They *have* left some traces today. Notably, the term "listserv" for an electronic mailing list, still occasionally used today, derives from an email reflector used on BITNET.

FTP, Gopher, and the forgotten pre-Web

The World Wide Web went from nowhere to ubiquity during a few short years in the early 1990s. Before that, from 1971 onwards, file transfer between Internet sites was normally done with a tool named ftp, for the File Transfer Protocol it used. Every hacker once knew how to use this tool.

Eventually ftp was mostly subsumed by web browsers speaking the FTP protocol themselves. This is why you may occasionally still see URLs with an "ftp:" service prefix; this informs the browser that it should expect to speak to an FTP server rather than an HTTP/HTTPS server.

There was another. The same year (1991) that Tim Berners-Lee was inventing the World Wide Web, a group of hackers at the University of Minnesota devised "Gopher", a hypertext protocol that was menu-centric rather than link-centric. For a few years Gopher competed vigorously with the early Web, and many hackers used both. Adoption was stalled when the University decided to charge a license fee for its implementation in early 1993. Then, early Web browsers added the ability to speak Gopher protocol and display Gopher documents. By 2000 Gopher was effectively dead, although a few Gopher servers are still operated in a spirit of nostalgia and irony.

Terminal confusion

The software terminal emulators on modern Unix systems are the near-end - and probably final - manifestations of a long and rather confused history. It began with early displays sometimes called "glass TTYs" because they emulated teletypes - but less expensively, because they didn't require consumables like paper. The phrase "dumb terminal" is equivalent. The first of these was shipped in 1969. The best-remembered of them is probably still the ADM-3 from 1975.

The very earliest VDTs, like the ASR-33 before them, could form only upper-case letters. An interesting hangover from these devices was that, even though most VDTs made after 1975 could form lower-case letters, Unix (and Linux as late as 2018) responded to a

login beginning with an upper-case letter by switching to a mode which upcased all input. If you create an account with this sort of login name and a mixed-case password, hilarity ensues. If the password is upper-case the hilarity is less desperate but still confusing for the user.

The classic "smart terminal" VDT designs that have left a mark on later computing appeared during a relatively short period beginning in 1975. Devices like the Lear-Siegler ADM-3A (1976) and the DEC VT-100 (1978) inherited the 80-character line width of punched cards (longer than the 72-character line length of teletypes) and supported as many lines as could fit on an approximately 4:3 screen (and in 2K bytes of display memory); they are the reason your software terminal emulator has a 24x80 or 25x80 default size.

These terminals were called "smart" because they could interpret control codes to do things like addressing the cursor to any point on the screen in order to produce truly 2-dimensional displays [7]. The ability to do bold, underline or reverse-video highlighting also rapidly became common. Colored text and backgrounds, however, only became available a few years before VDTs were obsolesced; before that displays were monochromatic. Some had crude, low-resolution dot graphics; a few types supported black-and-white vector graphics.

Early VDTs used a crazy variety of control codes. One of the principal relics of this era is the Unix terminfo database, which tracked these codes so terminal-using applications could do abstracted operations like "move the cursor" without being restricted to working with just one terminal type. The curses(3) library still used with software terminal emulators was originally intended to make this sort of thing easier.

After 1979 there was an ANSI standard for terminal control codes, based on the DEC VT-100 (being supported in the IBM PC's original screen driver gave it a boost) [8]. By the early 1990s ANSI conformance was close to universal in VDTs, which is why that's what your software terminal emulator does.

This whole technology category was rapidly wiped out in general-purpose computing, like dinosaurs after the Alvarez strike, when bit-mapped color displays on personal computers that could match the dot pitch of a monochrome VDT became relatively inexpensive, around 1992. The legacy VDT hardware lingered longest in dedicated point-of-sale systems, remaining not uncommon until as late as 2010 or so.

It's not true, as is sometimes suggested, that heritage from the VDT era explains the Unix command line - that actually predated VDTs, going back to the last generation of printing terminals in the late 1960s and early 1970s. Every hacker once knew that this is why we often speak of "printing" output when we mean sending it to standard output that is normally connected to a terminal emulator.

What the VDT era **does** explain is some of our heritage games (see next section) and a few surviving utility programs like vi(1), top(1) and mutt(1). These are what advanced visual interfaces looked like in the VDT era, before bitmapped displays and GUIs. This is why program interfaces that are two-dimensional but use characters only are now called TUI ("terminal user interface"), but the term is an anachronism; it was coined after "GUI" became common.

It also explains "screensavers". Every hacker once knew that these are so called because cathode-ray tubes were subject to a phenomenon called "phosphor burn-in" that could permanently damage the screen's ability to display information. Software to

randomly vary the image(s) on your display was originally written to prevent burn-in. Flatscreens don't have this problem; the secondary purpose of doing something visually interesting with that blank space took over.

The early, awful days of bitmapped displays

Terminals - which could usually only display a fixed alphabet of formed characters - were eventually replaced by the "bitmapped" style of display we're used to today, with individual screen pixels manipulable. Every hacker once knew that though there were earlier precedents done as research, the first production system with a bitmapped display was the Alto, built at the Xerox Palo Alto Research Center in 1973. (The laser printer and Ethernet were also invented there.)

It was **not** generally known that the Alto had a display only 608 pixels wide and 808 high - folk memory confused it with later displays in the 1024x1024 range. Its 1981 successor the Dandelion achieved 1024×809; an attempt to commercialize it as the "Xerox Star" failed. But in 1982 the newly-born Sun Microsystems shipped the Sun-1 with 1024×800 pixels; descendants of this machine (and very similar designs by other vendors) became enormously successful - until they were wiped out by descendants of the IBM PC in the late 1990s. Before that, most hackers dreamed of owning a Sun-class workstation, and the bitmapped display was a significant part of that allure.

Alas, these machines were too expensive for individuals. But from 1975 onwards primitive bitmapped-display began to appear on personal color computers such as the Apple II, often through consumer-grade television sets. These had one feature the workstation displays of the time lacked - color - but their pixel resolutions were laughable by comparison. The Apple II in "Hi-res" mode could only manage 280×192, an area you could easily cover with the palm of your hand on the displays of 2017.

People who remember these early consumer-grade color displays show a tendency to both think they arrived sooner than they did and filter their memories through a nostalgic haze. In reality they remained astoundingly bad compared to even a Sun-1 for a long time after the Apple II; you couldn't get even near the crispness of an 80×24 display of text on a VDT with them. Using a consumer-grade TV also meant coping with eyestrain-inducing artifacts around the limited amount of text it could display, due to NTSC chroma-luminance crossover.

Because hackers **needed** good text display for programming - and because many knew what workstation graphics in the 1024×1024 range looked like - many of us dismissed these displays (and the computers they were attached to) as near-useless toys. We sought workstation graphics when we could get it and used VDTs when we couldn't.

In 1984, the original Macintosh was the first consumer-grade machine with a dedicated bit-mapped display that come within even distant hail of workstation-class graphics - 512×342 black and white. It was shortly followed by the IBM EGA adapter and its numerous cheap clones with 640x350 at a whopping 16 colors, accompanied by flicker and other artifacts. In the following years, 256 colors gradually crept in, starting with a resolution of - wait for it - 320x200.

For another five years or so consumer hardware was still split between low-resolution b&w displays and *abysmally* low-resolution, hard-to-use color displays. If we had to use a consumer-grade computer at all, most of us tended to prefer the many b&w displays that

were relatively inexpensive, surprisingly nice on the eyes and better for our life's work - code.

There was, however, a cohort of hackers (especially among the younger and newer ones of the mid-1980s) who worked hard to get as much performance as they could from the pathetic color displays and TVs, as well as from the earliest attempts at computer sound hardware. The resulting chunky graphics with pixelated "sprites" moving across the screen, accompanied by a chirpy/buzzy style of music and sound effects is still used in some 21st-century games for pure nostalgic value. If you hear a reference to "8-bit" graphics or sound, that's what it means in these latter days.

Monitors capable of 1024×1024 *color* display did not reach even the high end of the consumer market until about 1990 and did not become generally available until about 1992. Not by coincidence, this was when PC manufacturers began to put serious pressure on workstation vendors. But for some years after that the quality of these displays remained relatively poor, with coarse dot pitches and fringing effects due to the expense and difficulty of manufacturing decent color masks for cathode-ray tubes. These problems weren't fully solved until CRTs were on the verge of being obsolesced by flatscreens.

Once we were able to take color displays at 1024x1024 and up for granted, a lot of this history faded from memory. Even people who lived through it have a tendency to forget what conditions were like before flatscreens, and are often surprised to be reminded how low the pixel resolutions were on older hardware.

Games before GUIs

Before bit-mapped color displays became common and made graphics-intensive games the norm, there was a vigorous tradition of games that required only textual interfaces or the character-cell graphics on a VDT.

These VDT games often found their way to early microcomputers as well. In part this was because some of those early micros themselves had weak or nonexistent graphical capabilities, and in part because textual games were relatively easy to port and featured as type-in projects in magazines and books.

The oldest group of games that were once common knowledge are the Trek family, a clade of games going back to 1971 in which the player flew the starship Enterprise through the Federation fighting Klingons and Romulans and other enemies. Every hacker over a certain age remembers spending hours playing these.

The history of the Trek clade is too complex to summarize here. The thing to notice about them is that the extremely crude interface (designed not even for VDTs but for teletypes!) hid what was actually a relatively sophisticated wargame in which initiative, tactical surprise, and logistics all played significant roles.

Every hacker once knew what the phrase "You are in a maze of twisty little passages, all alike" meant, and often used variants about confusing situations in real life (For example, "You are in a maze of twisty little technical standards, all different"). It was from the *very first* dungeon-crawling adventure game, *Colossal Cave Adventure* (1977). People who knew this game from its beginnings often thought of it as ADVENT, after its 6-character filename on the PDP-10 where it first ran [9].

When the original author of ADVENT wasn't inventing an entire genre of computer games, he was writing the low-level firmware for some of the earliest ARPANET routers. This was not generally known at the time, but illustrates how intimate the connection between these early games and the cutting edge of serious programming was. "Game designer" was not yet a separate thing, then.

You might occasionally encounter "xyzzy" as a nonce variable name. Every hacker used to know that xyzzy was a magic word in ADVENT.

ADVENT had a direct successor that was even more popular - Zork, first released in 1979 by hackers at MIT on a PDP-10 (initially under the name "Dungeon") and later successfully commercialized. This game is why every hacker once knew that a zorkmid was the currency of the Great Underground Empire, and that if you wander around in dark places without your lantern lit you might be eaten by a grue.

There was another family of games that took a different, more visual approach to dungeon-crawling. They are generally called "roguelikes", after the earliest widely-distributed games in this group, *Rogue* from 1980. They featured top-down, maplike views of dungeon levels through which the player would wander battling monsters and seeking treasure.

The most widely played games in this group were Hack (1982) and Nethack (1987). Nethack is notable for having been one of the earliest programs in which the development group was consciously organized as a distributed collaboration over the Internet; at the time, this was a sufficiently novel idea to be advertised in the project's name.

Rogue's descendants were the most popular and successful TUI games ever. Though they gradually passed out of universal common knowledge after the mid-1990s, they retain devoted minority followings even today. Their fans accurately point out that the primitive state of interface design encouraged concentration on plot and story values, leading to a surprisingly rich imaginative experience.

ASCII

ASCII, the American Standard Code for Information Interchange, evolved in the early 1960s out of a family of character codes used on teletypes.

ASCII, unlike a lot of other early character encodings, is likely to live forever - because by design the low 127 code points of Unicode **are** ASCII. If you know what UTF-8 is (and you should) every ASCII file is correct UTF-8 as well.

The following table describes ASCII-1967, the version in use today. This is the 16x4 format given in most references.

```
Dec Hex    Dec Hex    Dec Hex  Dec Hex  Dec Hex  Dec Hex   Dec Hex   Dec Hex
  0 00 NUL  16 10 DLE  32 20    48 30 0  64 40 @  80 50 P   96 60 `  112 70 p
  1 01 SOH  17 11 DC1  33 21 !  49 31 1  65 41 A  81 51 Q   97 61 a  113 71 q
  2 02 STX  18 12 DC2  34 22 "  50 32 2  66 42 B  82 52 R   98 62 b  114 72 r
  3 03 ETX  19 13 DC3  35 23 #  51 33 3  67 43 C  83 53 S   99 63 c  115 73 s
  4 04 EOT  20 14 DC4  36 24 $  52 34 4  68 44 D  84 54 T  100 64 d  116 74 t
```

```
 5 05 ENQ  21 15 NAK  37 25 %  53 35 5  69 45 E  85 55 U  101 65 e  117 75 u
 6 06 ACK  22 16 SYN  38 26 &  54 36 6  70 46 F  86 56 V  102 66 f  118 76 v
 7 07 BEL  23 17 ETB  39 27 '  55 37 7  71 47 G  87 57 W  103 67 g  119 77 w
 8 08 BS   24 18 CAN  40 28 (  56 38 8  72 48 H  88 58 X  104 68 h  120 78 x
 9 09 HT   25 19 EM   41 29 )  57 39 9  73 49 I  89 59 Y  105 69 i  121 79 y
10 0A LF   26 1A SUB  42 2A *  58 3A :  74 4A J  90 5A Z  106 6A j  122 7A z
11 0B VT   27 1B ESC  43 2B +  59 3B ;  75 4B K  91 5B [  107 6B k  123 7B {
12 0C FF   28 1C FS   44 2C ,  60 3C <  76 4C L  92 5C \  108 6C l  124 7C |
13 0D CR   29 1D GS   45 2D -  61 3D =  77 4D M  93 5D ]  109 6D m  125 7D }
14 0E SO   30 1E RS   46 2E .  62 3E >  78 4E N  94 5E ^  110 6E n  126 7E ~
15 0F SI   31 1F US   47 2F /  63 3F ?  79 4F O  95 5F _  111 6F o  127 7F DEL
```

However, this format - less used because the shape is inconvenient - probably does more to explain the encoding:

```
0000000 NUL    0100000      1000000 @    1100000 `
0000001 SOH    0100001 !    1000001 A    1100001 a
0000010 STX    0100010 "    1000010 B    1100010 b
0000011 ETX    0100011 #    1000011 C    1100011 c
0000100 EOT    0100100 $    1000100 D    1100100 d
0000101 ENQ    0100101 %    1000101 E    1100101 e
0000110 ACK    0100110 &    1000110 F    1100110 f
0000111 BEL    0100111 '    1000111 G    1100111 g
0001000 BS     0101000 (    1001000 H    1101000 h
0001001 HT     0101001 )    1001001 I    1101001 i
0001010 LF     0101010 *    1001010 J    1101010 j
0001011 VT     0101011 +    1001011 K    1101011 k
0001100 FF     0101100 ,    1001100 L    1101100 l
0001101 CR     0101101 -    1001101 M    1101101 m
0001110 SO     0101110 .    1001110 N    1101110 n
0001111 SI     0101111 /    1001111 O    1101111 o
0010000 DLE    0110000 0    1010000 P    1110000 p
0010001 DC1    0110001 1    1010001 Q    1110001 q
0010010 DC2    0110010 2    1010010 R    1110010 r
0010011 DC3    0110011 3    1010011 S    1110011 s
0010100 DC4    0110100 4    1010100 T    1110100 t
0010101 NAK    0110101 5    1010101 U    1110101 u
0010110 SYN    0110110 6    1010110 V    1110110 v
0010111 ETB    0110111 7    1010111 W    1110111 w
```

```
0011000 CAN    0111000 8    1011000 X    1111000 x
0011001 EM     0111001 9    1011001 Y    1111001 y
0011010 SUB    0111010 :    1011010 Z    1111010 z
0011011 ESC    0111011 ;    1011011 [    1111011 {
0011100 FS     0111100 <    1011100 \    1111100 |
0011101 GS     0111101 =    1011101 ]    1111101 }
0011110 RS     0111110 >    1011110 ^    1111110 ~
0011111 US     0111111 ?    1011111 _    1111111 DEL
```

Using the second table, it's easier to understand a couple of things:

- The Control modifier on your keyboard basically clears the top three bits of whatever character you type, leaving the bottom five and mapping it to the 0..31 range. So, for example, Ctrl-SPACE, Ctrl-@, and Ctrl-` all mean the same thing: NUL.
- Very old keyboards used to do Shift just by toggling the 32 or 16 bit, depending on the key; this is why the relationship between small and capital letters in ASCII is so regular, and the relationship between numbers and symbols, and some pairs of symbols, is sort of regular if you squint at it. The ASR-33, which was an all-uppercase terminal, even let you generate some punctuation characters it didn't have keys for by shifting the 16 bit; thus, for example, Shift-K (0x4B) became a [(0x5B)

It used to be common knowledge that the original 1963 ASCII had been slightly different. It lacked tilde and vertical bar; 5E was an up-arrow rather than a caret, and 5F was a left arrow rather than underscore. Some early adopters (notably DEC) held to the 1963 version.

If you learned your chops after 1990 or so, the mysterious part of this is likely the control characters, code points 0-31. You probably know that C uses NUL as a string terminator. Others, notably LF = Line Feed and HT = Horizontal Tab, show up in plain text. But what about the rest?

Many of these are remnants from teletype protocols that have either been dead for a very long time or, if still live, are completely unknown in computing circles. A few had conventional meanings that were half-forgotten even before Internet times. A **very** few are still used in binary data protocols today.

Here's a tour of the meanings these had in older computing, or retain today. If you feel an urge to send me more, remember that the emphasis here is on what was *common knowledge* back in the day. If I don't know it now, we probably didn't generally know it then.

NUL (Null) = Ctrl-@
: Survives as the string terminator in C.

SOH (Start of Heading) = Ctrl-A
: Rarely used (as Ctrl-A) as a section divider in otherwise textual formats. Some versions of Unix mailbox format used it as a message divider. One very old version-control system (SCCS) did something similar.

STX (Start of Text), ETX (End of Text) = Ctrl-B, Ctrl-C

Very rarely used as packet or control-sequence delimiters. You will probably never see this, and the only place I've ever seen it was on a non-Unix OS in the early 1980s. ETX is Ctrl-C, which is a SIGINT interrupt character on Unix systems, but that has nothing to do with its ASCII meaning per se and probably derives from abbreviating the word "Cancel".

EOT (End of Transmission) = Ctrl-D

As Ctrl-D, the way you type "End of file" to a Unix terminal.

ENQ (Enquiry) = Ctrl-E

In the days of hardware serial terminals, there was a convention that if a computer sent ENQ to a terminal, it should answer back with terminal type identification. While this was not universal, it at least gave computers a fighting chance of autoconfiguring what capabilities it could assume the terminal to have. Further back, on teletypes, the answerback had been a station ID rather than a device type; as late as the 1970s it was still generally remembered that ENQ's earliest name in ASCII had been WRU ("Who are you?").

ACK (Acknowledge) = Ctrl-F

It used to be common for wire protocols written in ASCII to use ENQ/ACK as a handshake, sometimes with NAK as a failure indication (the XMODEM/YMODEM/ZMODEM protocol did this [10]). Hackers used to use ACK in speech as "I hear you" and were a bit put out when this convention was disrupted in the 1980s by Bill The Cat's "Ack! Thppt!"

BEL (Bell) = Ctrl-G

Make the bell ring on the teletype - an attention signal. This often worked on VDTs as well, but is no longer reliably the default on software terminal emulators. Some map it to a visual indication like flashing the title bar.

BS (Backspace) = Ctrl-H

Still does what it says on the tin, though there has been some historical confusion over whether the backspace key on a keyboard should behave like BS (nondestructive cursor move) or DEL (backspace and delete). Never used in textual data protocols.

HT (Horizontal tab) = Ctrl-I

Still does what it says on the tin. Sometimes used as a field separator in Unix textual file formats, but this is now old-fashioned and declining in usage.

LF (Line Feed) = Ctrl-J

The Unix textual end-of-line. Printing terminals interpreted it as "scroll down one line"; the Unix tty driver would normally wedge in a CR right before it on output (or in early versions, right after).

VT (Vertical Tab) = Ctrl-K

In the days of printing terminals this often caused them to scroll down a configurable number of lines. VDTs had any number of possible behaviors; at least some pre-ANSI ones interpreted VT as "scroll **up** one line". The only reason anybody remembers this one at all is that it persisted in Unix definitions of what a whitespace character is, even though it's now extinct in the wild.

FF (Form Feed) = Ctrl-L

Eject the current page from your printing terminal. Many VDTs interpreted this as a "clear screen" instruction. Software terminal emulators sometimes still do. Often interpreted as a "screen refresh" request in textual-input Unix programs that bind other control characters (shells, editors, more/less, etc)

CR (Carriage Return) = Ctrl-M

It is now possible that the reader has never seen a typewriter, so this needs explanation: "carriage return" is the operation of moving your print head or cursor to the left margin. Windows, other non-Unix operating systems, and some Internet protocols (such as SMTP) tend to use CR-LF as a line terminator, rather than bare LF. The reason it was CR-LF rather than LF-CR goes back to Teletypes: a Teletype printed ten characters per second, but the print-head carriage took longer than a tenth of a second to return to the left side of the paper. So if you ended a line with line-feed, then carriage-return, you would usually see the first character of the next line smeared across the middle of the paper, having been struck while the carriage was still zipping to the left. Pre-Unix MacOS used a bare CR.

SO (Shift Out), SI (Shift In) = Ctrl-N, Ctrl-O

Escapes to and from an alternate character set. Unix software used to emit them to drive pre-ANSI VDTs that interpreted them that way, but native Unix usage is rare to nonexistent. On teletypes with a two-color ink ribbon (the second color usually being red) SO was a command to shift to the alternate color, SI to shift back.

DLE (Data Link Escape) = Ctrl-P

Sometimes used as a packet-framing character in binary protocols. That is, a packet starts with a DLE, ends with a DLE, and if one of the interior data bytes matches DLE it is doubled.

DC[1234] (Device Control [1234]) = Ctrl-[QRST]

Four device control codes used by the once-ubiquitous ASR-33 teletype to turn on and off its paper-tape reader and punch, which were used to read and write machine-readable data. On serial terminals, there was a common software flow-control protocol - used over ASCII but separate from it - in which XOFF (DC3) was used as a request to pause transmission and XON (DC1) was used as a request to resume transmission. As Ctrl-S and Ctrl-Q these were implemented in the Unix terminal driver and long outlived their origin in the Model 33 Teletype. And not just Unix; this was implemented in CP/M and DOS, too.

NAK (Negative Acknowledge) = Ctrl-U

See the discussion of ACK above.

SYN (Synchronous Idle) = Ctrl-V

Never to my knowledge used specially after teletypes, except in synchronous serial protocols never used on micros or minis. Be careful not to confuse this with the SYN (synchronization) packet used in TCP/IP's SYN SYN-ACK initialization sequence. In an unrelated usage, many Unix tty drivers use this (as Ctrl-V) for the literal-next character that lets you quote following control characters such as Ctrl-C.

ETB (End of Transmission Block) = Ctrl-W

Nowadays this is usually "kill window" on a web browser, but it used to mean "delete previous word" in some contexts and sometimes still does.

CAN (Cancel), EM (End of Medium) = Ctrl-X, Ctrl-Y
: Never to my knowledge used specially after teletypes.

SUB (Substitute) = Ctrl-Z
: DOS and Windows use Ctrl-Z (SUB) as an end-of-file character; this is unrelated to its ASCII meaning. It was common knowledge then that this use of ^Z had been inherited from a now largely forgotten earlier OS called CP/M (1974), and into CP/M from earlier DEC minicomputer OSes such as RSX-11 (1972). Unix uses Ctrl-Z as the "suspend process" command keystroke.

ESC (Escape)
: Still commonly used as a control-sequence introducer. This usage is especially associated with the control sequences recognized by VT100 and ANSI-standard VDTs, and today by essentially all software terminal emulators

[FGRU]S ({Field|Group|Record|Unit} Separator)
: There are some uses of these in ATM and bank protocols (these have never been common knowledge, but I'm adding this note to forestall yet more repetitions from area specialists who will apparently otherwise keep telling me about it until the end of time). FS, as Ctrl-\, sends SIGQUIT under some Unixes, but this has nothing to do with ASCII. Ctrl-] (GS) is the exit character from telnet, but this also has nothing to do with its ASCII meaning.

DEL (Delete)
: Usually an input character meaning "backspace and delete". Under older Unix variants, sometimes a SIGINT interrupt character.

The "Break" key on a teletype mimicked the current-loop line condition of a broken wire. A quiet teletype machine would start cycling but doing nothing when the line was disconnected, so it was possible to know that this had occurred. (It was effectively receiving a series of NULL characters.)

Not all of these were so well known that any hacker could instantly map from mnemonic to binary, or vice-versa. The well-known set was roughly NUL, BEL, BS, HT, LF, FF, CR, ESC, and DEL.

There are a few other bits of ASCII lore worth keeping in mind...

- A Meta or Alt key on a VDT added 128 to the ASCII keycode for whatever it's modifying (probably - on a few machines with peculiar word lengths they did different things). Software terminal emulators have more variable behavior; many of them now simply insert an ESC before the modified key, which Emacs treats as equivalent to adding 128.
- An item of once-common knowledge that was half-forgotten fairly early (like, soon after VDTs replaced teletypes) is that the binary value of DEL (0x7F, 0b01111111) descends from its use on paper tape. Seven punches could overwrite any character in ASCII, and tape readers skipped DEL (and NUL - no punches). This is why DEL was anciently called the "Rubout" character and is an island at the other end of the ASCII table from the other control characters.
- VDT keyboards often had a "Break" key inherited from the ASR-33 (there's a vestigial remnant of this even on the IBM PC keyboard). On the AS-33 this had mimicked the current-loop line condition of a broken wire, which was detectable. On a VDT

this didn't send a well-formed ASCII character; rather, it caused an out-of-band condition that would be seen as a NUL with a framing error at the other end. This was used as an attention or interrupt signal [11].

You can study the bit structure of ASCII using ascii(1). Both of the tables above were generated using it.

The slow birth of distributed collaboration

Nowadays we take for granted a public infrastructure of distributed version control and a lot of practices for distributed teamwork that go with it - including development teams that never physically have to meet. But these tools, and awareness of how to use them, were a long time developing. They replace whole layers of earlier practices that were once general but are now half- or entirely forgotten.

The earliest practice I can identify that was directly ancestral was the DECUS tapes. DECUS was the Digital Equipment Corporation Users' Group, chartered in 1961. One of its principal activities was circulating magnetic tapes of public-domain software shared by DEC users. The early history of these tapes is not well-documented, but the habit was well in place by 1976.

One trace of the DECUS tapes seems to be the README convention. While it entered the Unix world through USENET in the early 1980s, it seems to have spread there from DECUS tapes. The DECUS tapes begat the USENET source-code groups, which were the incubator of the practices that later became "open source". Unix hackers used to watch for interesting new stuff on comp.sources.unix as automatically as they drank their morning coffee.

The DECUS tapes and the USENET sources groups were more of a publishing channel than a collaboration medium, though. Three pieces were missing to fully support that: version control, patching, and forges.

Version control was born in 1972, though SCCS (Source Code Control System) didn't escape Bell Labs until 1977. The proprietary licensing of SCCS slowed its uptake; one response was the freely reusable RCS (Revision Control System) in 1982.

The first real step towards across-network collaboration was the patch(1) utility in 1984. The concept seems so obvious now that even hackers who predate patch(1) have trouble remembering what it was like when we only knew how to pass around source-code changes as entire altered files. But that's how it was.

Even with SCCS/RCS/patch the friction costs of distributed development over the Internet were still so high that some years passed before anyone thought to try it seriously. I have looked for, but not found, definite examples earlier than nethack. This was a roguelike game launched in 1987. Nethack developers passed around whole files - and later patches - by email, sometimes using SCCS or RCS to manage local copies. [12].

Distributed development could not really get going until the third major step in version control. That was CVS (Concurrent Version System) in 1990, the oldest VCS still in wide use at time of writing in 2017. Though obsolete and now half-forgotten, CVS was the first version-control system to become so ubiquitous that every hacker once knew it. CVS, however, had significant design flaws [13]; it fell out of use rapidly when better alternatives became available.

Between around 1989 and the breakout of mass-market Internet in 1993-1994, fast Internet became available enough to hackers that distributed development in the modern style began to become thinkable. The next major steps were not technical changes but cultural ones.

In 1991 Linus Torvalds announced Linux as a distributed collaborative effort. It is now easy to forget that early Linux development used the same patch-by-email method as nethack - there were no public Linux repositories yet. The idea that there **ought** to be public repositories as a normal practice for major projects (in addition to shipping source tarballs) wouldn't really take hold until after I published "The Cathedral and the Bazaar" in 1997. While CatB was influential in promoting distributed development via shared public repositories, the technical weaknesses of CVS were in hindsight probably an equally important reason this practice did not become established sooner and faster.

The first dedicated software forge was not spun up until 1999. That was SourceForge, still extant today (2018). At first it supported only CVS, but it sped up the adoption of the (greatly superior) Subversion, launched in 2000 by a group of former CVS developers.

Between 2000 and 2005 Subversion became ubiquitous common knowledge. But in 2005 Linus Torvalds invented git, which would fairly rapidly obsolesce all previous version-control systems and is a thing every hacker **now** knows [14].

Key dates

These are dates that every hacker knew were important at the time, or shortly afterwards. I've tried to concentrate on milestones for which the date - or the milestone itself - seems to have later passed out of folk memory.

1961
: MIT takes delivery of a PDP-1. The first recognizable ancestor of the hacker culture of today rapidly coalesces around it.

1969
: Ken Thompson begins work on what will become Unix. First commercial VDT ships; it's a glass TTY. First packets exchanged on the ARPANET, the direct ancestor of today's Internet.

1970
: DEC PDP-11 first ships; architectural descendants of this machine, including later Intel microprocessors, will come to dominate computing.

1973
: Interdata 32 ships; the long 32-bit era begins [15]. Unix Edition 5 (not yet on the Interdata) escapes Bell Labs to take root at a number of educational institutions. The XEROX Alto pioneers the "workstation" - a networked personal computer with a high-resolution display and a mouse.

1974

CP/M first ships; this will be the OS for a large range of microcomputers until effectively wiped out by MS-DOS after 1981. MS/DOS will, however, have been largely cloned from CP/M; this theft leaves rather unmistakable traces in the BIOS. It's also why MS-DOS has filenames with at most 8 characters of name and 3 of extension.

1975

First Altair 8800 ships; beginning of heroic age of microcomputers. First 24x80 and 25x80 "smart" (addressable-cursor) VDTs. ARPANET declared "operational", begins to spread to major universities.

1976

"Lions' Commentary on UNIX 6th Edition, with Source Code" released. First look into the Unix kernel source for most hackers, and was a huge deal in those pre-open-source days. First version of ADVENT is written. First version of the Emacs text editor.

1977

Unix ported to the Interdata; first version with a kernel written largely in C rather than machine-dependent assembler. Second generation of home computers (Apple II and TRS-80 Model 1) ship. SCCS, the first version-control system, is publicly released. Colossal Cave Adventure ships.

1978

First BBS launched - CBBS, in Chicago.

1979

MIT Dungeon, later known as Zork, is written; the first grues lurk in dark places.

1980

Rogue, ancestral to all later top-view dungeon-crawling games, is invented. USENET begins.

1981

First IBM PC ships; end of the heroic age of micros. TCP/IP is implemented on a VAX-11/780 under 4.1BSD Unix; ARPANET and Unix cultures begin to merge.

1982

After some false starts from 1980-1981 with earlier 68000-based micros of similar design, the era of commercial Unix workstations truly begins with the founding and early success of Sun Microsystems. RCS, the second version-control system, ships.

1983

PDP-10 canceled; this is effectively the end of 36-bit architectures anywhere outside of deep mainframe country, though Symbolics Lisp machines hold out a while longer. ARPANET, undergoing some significant technical changes, becomes Internet.

1984

AT&T begins a largely botched attempt to commercialize Unix, clamping down on access to source code. In the BBS world, FidoNet is invented. The patch(1) utility is invented.

1985

RMS published GNU Manifesto. This is also roughly the year the C language became the dominant lingua franca of both systems and applications programming, eventually displacing earlier compiled language so completely that they are almost forgotten. First Model M keyboard ships.

1986

Intel 386 ships; end of the line for 8- and 16-bit PCs. Consumer-grade hardware in this class wouldn't be generally available until around 1989, but after that would rapidly surpass earlier 32-bit minicomputers and workstations in capability.

1987

USENET undergoes the Great Renaming. Perl, first of the modern scripting languages, is invented.

1991

Linux and the World Wide Web are (separately) launched. Python scripting language invented.

1992

Bit-mapped color displays with a dot pitch matching that of a monochrome VDT (and a matching ability to display crisp text at 80x25) ship on consumer-grade PCs. Bottom falls out of the VDT market.

1993

Linux gets TCP/IP capability, moves from hobbyist's toy to serious OS. America OnLine offers USENET access to its users; "September That Never Ended" begins. Mosaic adds graphics and image capability to the World Wide Web.

1994

Mass-market Internet takes off in the U.S. USB promulgated.

1995-1996

Peak years of UUCP/USENET and the BBS culture, then collapse under pressure from mass-market Internet.

1997

I first give the "Cathedral and Bazaar" talk.

1999

Banner year of the dot-com bubble. End of workstation era: Market for Suns and other proprietary Unix workstations collapses under pressure from Linux running on PCs. Launch of SourceForge, the first public shared-repository site.

2000

Subversion first ships.

2001

Dot-com bubble pops. PC hardware with workstation-class capabilities becomes fully commoditized; pace of visible innovation in mass-market computers slows noticeably.

2005
: Major manufacturers cease production of cathode-ray tubes in favor of flat-panel displays. Flat-panels have been ubiquitous on new hardware since about 2003. There is a brief window until about 2007 during which high-end CRTs no longer in production still exceed the resolution of flat-panel displays and are still sought after. Also in 2005, AOL drops USENET support and Endless September ends. Git first ships.

2007-2008
: 64-bit transition in mass market PCs; the 32-bit era ends. Single-processor speeds plateau at 4±0.25GHz. iPhone and Android (both with Unix-based OSes) first ship.

Request to contributors

A lot of people reading this have been seized by the urge to send me some bit of lore or trivia for inclusion. Thank you, but bear in mind that the most important choice is what to leave out. Here are some guidelines:

- I'm trying to describe *common knowledge at the time*. That means not every bit of fascinating but obscure trivia belongs here.
- Anything from a tech generation before early minis - in particular the era of mainframes, punched cards, and paper tape - is out of scope. I gotta draw the line somewhere, and it's there.
- Stories about isolated survivals of old tech today are not interesting if the tech wasn't once common knowledge.
- Please do not send me timeline entries for dates which you think are important unless you think the date has generally been forgotten, or is in serious danger of same.

Supporting this work

If you enjoyed this, you should probably be part of the Loadsharers network. Give generously; the civilization you save could be your own.

Related Reading

How To Become A Hacker

The Lost Art of C Structure Packing

Change history

1.0: 2017-01-26
: Initial version.

1.1: 2017-01-27
: Pin down the date DB-9 came in. Added a minor section on the persistence of octal. More on the afterlife of RS-232.

1.2: 2017-01-29
: More about the persistence of octal. Mention current-loop ASR-33s. 36-bit machines and their lingering influence. Explain ASCII shift. A bit more about ASCII-1963. Some error correction.

1.3: 2017-01-30
: Added "Key dates" and "Request to contributors".

1.4: 2017-02-03
: The curious survival of the Hayes AT command set.

1.5: 2017-02-04
: TTL in serial and maker devices. The AT Hayes prefix explained. UUCP and long distance rates. Reference to space-cadet keyboard removed, as it turned out to ship a 32-bit word. Improved description of ASCII shift.

1.6: 2017-02-08
: How VDTs explain some heritage programs, and how bitmapped displays eventually obsolesced them. Explain why the ADM-3 was called "dumb" even though it was smart.

1.7: 2017-02-09
: The BBS subculture. XMODEM/YMODEM/ZMODEM. Commercial timesharing. Two dates in USENET history.

1.8: 2017-02-14
: Heritage games. The legacy of all-uppercase terminals. Where README came from. What "core" is. The ARPANET. Monitoring your computer with a radio.

1.9: 2017-02-17
: DEL was once Rubout. The Trek games. XYZZY.

1.10: 2017-02-20
: The Break key. uuencode/uudecode. Why older Internet protocols only assume a 7-bit link. The original meanings of SO/SI. WRU and station ID on teletypes. BITNET and other pre-Internets.

1.11: 2017-03-02
: SIGHUP. Six-bit characters on 36-bit machines. XMODEM required 8 bits. Screensavers.

1.12: 2017-03-18
: Note just how crazily heterogenous the six-bit character sets were. FTP. Ctrl-V on Unix systems. A correction about uu{de|en}code. Timeline updates for '74 and '77.

1.13: 2017-04-09
: Null-modem cables. The term "TUI". Why it's CR-LF and not LF-CR. Timesharing.

1.14: 2017-07-18
: Report the actual range of human audibility. By popular demand, include a "vertical" 16x4 version of the ASCII table.

1.15: 2017-07-31
: The sad tale of Gopher. Added TOC. Slow birth of distributed development. Early history of bitmapped displays.

1.16: 2017-09-14
: "8-bit" graphics and sound. Control-W.

1.17: 2018-03-01
: The uppercasing-login "feature": it lives!

1.18: 2018-07-18
: Add link to a video explaining the beeping and whooshing sounds.

1.19: 2019-06-15
: Fix for a minor typo.

1.20: 2019-07-01
: Ctrl-W lives.

1.21: 2019-07-17
: So does Ctrl-L.

1.22: 2023-04-19
: Line-delimiter fun. A bit more explanation of DC1/DC3 and Break. Add ADVENT to the timeline.

---

1. Actually, there was an even older style of tty interface derived from telegraph circuits and called "current loop" that the ASR-33 originally used; in the 1970s dual-mode ASR-33s that could also speak RS-232 began to ship, and RS-232 eventually replaced current loop entirely.

2. A full explanation of the magic of the AT prefix can be found at http://esr.ibiblio.org/?p=7333&cpage=1#comment-1802568

3. Python 3, Perl 6, and Rust have at least gotten rid of the dangerous leading-0-for-octal syntax, but Go kept it

4. Early Intel microprocessors weren't much like the PDP-11, but the 80286 and later converged with it in important ways.

5. Except on teletypes, which used 7E2.

6. The old free-floating USENET still exists too, but Google Groups is where you can find what has been preserved of the historical USENET archives.

7. Confusingly, the ADM-3A (which could address any screen cell) was described in marketing copy as a "dumb" terminal, not a "smart" one. This is because there was a rival definition of "smart" as capable of doing local editing of the screen without involving the remote computer, like an IBM 3270. But since minicomputers never used that capability this definition was never live in the Unix world, and has mostly faded out of use.

8. It was not commonly known that the VT100 was designed to fit a 1976 standard called ECMA-48; ANSI simply adopted it.

9. There is a port of ADVENT in modern C.

10. In fact, the inventor reports: "I invented Xmodem based upon an ASCII pocket reference card. ACK, NAK, SOH, EOT, sounded like good *words* to use to make a protocol."

11. The Break key has a cognate in "Break!" used as a similar attention signal or interrupt in voice radio procedure. Both derive from 19th-century telegraphic practice.

12. I was an early nethack devteam member. I did not at the time understand how groundbreaking what we were doing actually was.

13. In particular, CVS tends to behave very badly around file renames and deletions.

14. I'm deliberately not covering distributed VCSes other than git here; while some of them were historically and technically interesting, none ever became common knowledge.

15. There were a few 32-bit minis before the Interdata, but they seem to have been designed for real-time or other non-timesharing uses.

Last updated 2023-04-19 18:40:27 EDT