Text−Terminal−HOWTO

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Text−Terminal−HOWTO

David S. Lawyer mailto:dave@lafn.org

v1.20, February 2001

This document explains what text terminals are, how they work, how to install and configure them, and provides some info on how to repair them. If you don't have a terminal manual, it may be of help. While it's written for real terminals on a Linux system, some of it is also applicable to terminal emulation and may be helpful for non−Linux systems.

---

6. Terminal Special Files such as /dev/tty

7. Some Details on How Terminals Work

8. Special Features of Some Terminals

9. Terminal Emulation; the Console

10. Flow Control (Handshaking)

14. Computer Set−Up (Configure) Details

15. Terminfo and Termcap (detailed)

16. Using the Terminal

17. Special Uses for a Terminal

18. Trouble−Shooting

19. Repair & Diagnose

20. Appendix A: General

21. Appendix B: Escape Sequence Commands Terminology

22. Appendix C: Serial Communications on EIA−232 (RS−232)

23. Appendix D: Notes by Brand Name

---

1. Introduction

For a quick attempt to install a terminal see Quick Install.

1.1 Copyright, Trademarks, Disclaimer, & Credits

Copyright

Disclaimer

While I haven't intentionally tried to mislead you, there are likely a number of errors in this document. Please let me know about them. Since this is free documentation, it should be obvious that I cannot be held legally responsible for any errors.

Trademarks.

Any brand names (starts with a capital letter) should be assumed to be a trademark). Such trademarks belong to their respective owners.

Credits

Much of the section "Physical Connection" is from Serial−HOWTO v. 1.11 (1997) by Greg Hankins (with his permission). His "How Do I Set Up A Terminal Connected To My PC?" was incorporated into v1.00 at various places. v1.09 has about 25 changes (and error corrections) suggested by Alessandro Rubini who reviewed this HOWTO. Jeremy Jon Spykerman told me about using a keyboardless terminal as a console for a monitorless PC (using ttysnoop).

1.2 Future Plans: You Can Help

Please let me know of any errors in facts, opinions, logic, spelling, grammar, clarity, links, etc. But first, if the date is over a few months old, check to see that you have the latest version. Please send me any info that you think belongs in this document.

Starting with version 1.00, a first attempt was made to help people set up terminals without recourse to a terminal manual. Much more is needed in this respect. One way to solve this problem would be if terminal manufacturers put their manuals on the Internet. I suggest that you encourage them to do so. The task of providing information on how to configure most terminals in this HOWTO is daunting. There are so many different terminals, but there are far fewer models than there used to be in the 1980,s so the task is not totally infeasible.

Please send me any surplus terminal manuals which you may have, especially on terminals made within the past 10 years (but I'll accept older ones also). Also, you might want to write up something on a certain terminal to put in the Appendix D: Notes by Brand Name.

1.3 New Versions of this HOWTO

New versions of the Text−Terminal−HOWTO should be released every month or two to browse and/or download at LDP mirror sites. For a list of mirror sites see: http://linuxdoc.org/mirrors.html. Various formats are available. If you only want to quickly check the date of the latest version look at http://linuxdoc.org/HOWTO/Text−Terminal−HOWTO.html. The version your are currently reading is: v1.20, February 2001 . New in this version: v1.20 Feb. 2001 Revised: Is hardware flow control done by hardware ?, Bash bugs (with terminals), stuck key, PCTerm could work w/Linux

1.4 Related HOWTOs, etc.

Go to the websites shown above to get these.

- Serial−HOWTO has info on Multiport Serial Cards used for both terminals and banks of modems. It has general technical info on the serial port including troubleshooting it.
- Low−Level Terminal Interface part of "GNU C Library Reference manual" (in libc (or glibc) docs package). It covers the detailed meaning of "stty" commands, etc.
- MacTerminal mini−HOWTO
- Modem−HOWTO

- Serial−Programming−HOWTO
- NC mini−HOWTO
- NCD−X−Terminal mini−HOWTO
- XDM−and−X−Terminal mini−HOWTO
- NCD−HOWTO
- Thinclient−HOWTO
- Xterminal−HOWTO (unmaintained). It's at http://sunsite.unc.edu/pub/Linux/docs/HOWTO/unmaintained/mini/Xterminal

1.5 Terminology Used in this Document

Configuration means the same as set−up. While Linux commands take options (using − symbols), options in a broader sense include various other types of choices. Install in the broad sense includes setting up (configuring) software and hardware. A statement that I suspect is true (but may not be) ends with 2 question marks: ?? If you know for sure, let me know.

1.6 What is a Terminal ?

A terminal consists of a screen and keyboard that one uses to communicate remotely with a (host) computer. One uses it just like it was a personal computer but the terminal is remote from its host computer (on the other side of the room or even on the other side of the world). Programs execute on the host computer but the results display on the terminal screen. Its computational ability is relatively low (otherwise it would be a computer and not a terminal). This computational ability is generally limited to the ability to display what is sent to it (possibly including full−screen graphics) and the ability to send to the host what is typed at the keyboard.

In the days of mainframes from the mid 1970's to the mid 1980's, most people used terminals to communicate with computers. They typed in programs, ran programs, wrote documents, issued printing commands, etc. A cable connected the terminal to the computer (often indirectly). It was called a terminal since it was located at the terminal end of this cable.

If you've been using Linux (except for X−Window use) with a computer monitor and keyboard you already know what a terminal is because you have been using one (or more precisely a "virtual terminal"). The monitor (along with the keyboard) is emulating a terminal. In X−Windows the programs xterm, rxvt, and zterm emulate terminals.

A real terminal is different from a monitor because it's a different electronic setup. A text terminal is often connected to a serial port of the computer via a long cable. Thus, in contrast to a monitor which is usually located right next to the computer, a terminal may be quite a distance away from its host computer. The video card inside a computer stores the video image seen on the monitor screen. For a terminal, the equivalent of this video card is built right into the terminal but since text terminals are often monochrome without much graphics, the capabilities of its "video card" are rather weak. Also, most text terminals do not have mice.

In network client−server terminology, one might think that the terminal is the client and that the host computer is the server. The terminal has been called a "thin client" by some. But it is not actually a "client" nor is the host a "server". The only "service" the host provides is to receive every letter typed at the keyboard and react to this just like a computer would. The terminal is like a window into the computer just like a monitor (and keyboard) are. You may have already used virtual terminals in Linux (by pressing Left Alt−F2, etc.). A real terminal is just like running such a virtual terminal but you run it on its own terminal screen instead of having to share the monitor screen. In contrast to using a virtual terminal at the console (monitor), this allows another person to sit at the real terminal and use the computer simultaneously with others.

2. Types of Terminals

2.1 Dumb Terminals

There are various conflicting definitions of "dumb terminal" but as time goes by, more and more terminals are called dumb. This document mainly covers text terminals which display only text on the screen. It might be titled "Dumb−Terminal−HOWTO" but in some magazines articles any terminal, no matter how smart, including ones which present a full graphical user interface (GUI), are called dumb. If all terminals are "dumb" then there is no point of prefixing the word "dumb" to terminal (except as a sales pitch to sell computers or the like in place of "smart" terminals). Due to the ambiguous meaning of "dumb terminal" it is not classified here as a type of terminal.

2.2 Text Terminals

For a text terminal, a 2−way flow of information between the computer and the terminal takes place over the cable that connects them together. This flow is in ASCII bytes where each byte usually represents a character. Bytes typed at the keyboard go to the computer and most bytes from the computer are displayed on the terminal screen. Special bytes (or sequences of bytes) from the computer tell the terminal where to move the cursor to, what to erase, where to begin and end underlining and/or blinking and/or bold, etc. There are often hundreds of such special commands and many terminals can even change fonts.

The communication uses characters (letters) encoded using a code chart for the character set being used. Usually, the first 128 bytes out of 256 possible bytes use ASCII codes. Terminals for Unix−like systems, normally connect to computers via a cable running between the asynchronous serial ports (RS−232−C = EIA−232−D) of the host computer and terminal. Sometimes the connection is via modem or terminal server, etc.

Other names for text terminals are "serial terminal", "character−cell terminal", "ASCII/ANSI terminal", "asynchronous terminal", "data terminal", "video terminal" and "video display terminal" (VDT) and "green terminal" (since many used green displays). In olden days "video display unit" (VDU) was used for terminals but strictly speaking, it excludes the keyboard.

"Block mode" was used exclusively by old IBM mainframe terminals but many modern terminals also have this capability (which is not used much). The characters you type are temporarily retained in the terminal memory (and may possibly be edited by a built−in editor at the terminal). Then when the send key (or the like) is pressed, a block of characters (sometimes just a line of characters) is sent to the computer all at once. Block mode (as of late 1998) is not supported by Linux. See section Block Mode.

2.3 Graphic Capabilities of Text Terminals

Many text terminals can display bit−mapped images, but not in color. Unfortunately, the popular image formats used on the Internet are not supported. The protocols for such graphics include: Tektronix Vector Graphics, ReGIS (DEC), Sixel (DEC), and NAPLPS (North American Presentation Level Protocol Syntax).

Even without bit−mapped images, ordinary text terminals can sort of display images. One may form arrows <−−− and draw boxes with _ and |. With special graphic character sets, even more is possible. By using all the letters, one may produce "ascii graphics" art. The term "graphics terminal" usually means a terminal that can display bit mapped images. However, this term is sometimes applied also to text−only terminals since text is a limited form of graphics.

Graphics displays

There are two basic types of graphics displays: raster and vector (rarely used). Raster graphics (bit−mapped) puts dots on the screen by horizontal scan lines drawn by an electron beam (or by activating pixels or dots on a flat screen). Vector graphic displays were intended to be used for monochrome screens that don't have any dots. They use smart electronics to draw lines and curves with an electron beam that can move in any direction (at any angle and location). True vector graphics draws high quality lines without zig−zags but is both rare and expensive. Raster graphics is almost universally used today for both PCs and text terminals. For PC's, images encoded in vector graphic format are sometimes used but they are translated to raster graphics format for display (with a drop in image quality).

2.4 Thin Clients (Terminals ?)

Introduction

These are thin (minimal) computers that behave something like terminals. Since text terminals (except for very old one) run an embedded operating system, they are also like a computer. Thin−clients need more computing power. In contrast to text−terminals thin clients all display a modern high−speed GUI. They are dependent on more powerful computers (servers) for their operation. For a true terminal, the computing work and disk storage will all be done on the server. At the other extreme, most of this work and storage is done at the client but some things such as administration, still depend on the server. Since such a client is not really "thin" it's a misnomer to call it a "thin client". Some claim that text−terminals are also thin clients but they are not really since they don't conform to the client−server model.

Thus a thin client is like a terminal and perhaps should be called a terminal. The thin client has a GUI with a mouse that makes it seem like you are using a computer. You are, but that computer may be far away and have many other people using it at the same time you are. Communication is over a high speed network cable, often twisted pair (UTP). Some thin clients can in addition emulate a text terminal and have a serial port connector for that purpose. One even has a USB interface.

There are two major types of thin clients (and some additional types as well which will not be discussed here). One type is the "Window Terminal" which usually runs under MS software and servers. The other type is the "network computer" which is supposed to be platform neutral. This implies they should work with both MS Windows and Linux but early models may not be easy to use with Linux.

Window terminals

These can run under MS Windows NT/2000 using a proprietary protocol. They are true terminals since all the computing work is done by the server running Windows. They are also called "Window−based Terminals" (WBT). Some have support for unix−like systems as well and may not claim to be WBTs (even though they can be used as WBTs). They are something like computers since they run an operating system (often stored in flash memory so it may be updated). Some can support X−Windows also and can be used for Linux (from a Linux server). Many so called "network computers" can also run X−Windows. This will be discussed in the next section.

For displaying the MS−Windows GUI, Citrix was (and is) a major player with what was called Winterm using it's WinFrame software (which supported Windows 3.1). Microsoft licensed some of this and then came out with Hydra (code name), also known as "Windows Terminal Server". It works with versions 4 or higher of MS Windows NT. Then other companies that had their own proprietary systems for MS Windows decided to support Microsoft's system.

Citrix uses its ICA protocol and has created an add−on to Hydra known as pICAsso so that WinFrame (ICA) based terminals can use the Hydra system. There exits a ICA client that internally runs Linux (but connects to a MS Window's server). Microsoft has substituted RDP (Remote Desktop Protocol) for ICA. Citrix has replaced WinFrame with MetaFrame which supports Windows 95, etc. and is used in conjunction with "Windows NT Terminal Server Edition".

The above is sometimes called "network computing" since the terminals and servers connect to each other over a network. Network computers may be somewhat different as described below.

Network computers (NCs)

These are neither true computers nor true terminals but are something in−between. One type of network computer (NC's) is a computer with a CPU but no hard Disk. The OS it needs to run is sent to it over a network. NCs are full−graphics and use the services of a server computer. They are a little different from terminals since some of the programs they run may execute on their own CPU chips. Running a browser was supposed to be one of their primary functions and thus Java code applets may be sent to them for execution. Many NCs support X−Windows so that one may use a Linux server to support it. IBM called their NC a "NetStation" but now calls it "NetVista". They should work on Intranet type networks and NetVista even runs the the Linux OS.

Wintel came out with a "NetPC" which, unlike the above, is almost a PC computer. However, it has no removable disks so users can't install their own software or obtain copies of anything.

Thin clients under Linux

Linux provides NFS (Network File System) so that if ordinary computers are connected to each other via a network, then a person on one computer can run programs on another computer. Such a program sends messages to you over the network so that it appears just like the program was being run by your local computer. It works also with X−Windows so that one may see GUI images generated on another computer. Linux also allows a computer to be diskless (see Diskless−HOWTO) and boot over a network. Thus using a diskless computer which runs NFS enables you to run programs on another computer (the server). This is just like using a NC (Network Computer). It's not really a NC but it's emulating a type of NC. It's also often called a "terminal" and in some sense it is.

Thus if you have an old PC with an ethernet card (NIC) you may be able to use it as a NC. The details of this are covered in Thinclient−HOWTO. Even if your old PC doesn't have a NIC, you could still use it as a text−terminal. See Terminal Emulation.

There are also a number of genuine Network Computers (NC) that will work with a Linux server. Today some NCs run the Linux OS inside the NC. Before Linux became popular, NCs didn't run the Linux OS but required some other OS. But even if the NC needs a non−linux OS, it's often possible to make it work with a Linux Server. The non−linux OS is simply stored as files on the Linux Server. Then when the NC starts up it sends a message to the Linux Server asking for the non−linux OS files. This non−linux OS is thus sent to the NC over the network and the NC boots.

The Linux Server runs the NFS and X−Windows both of which must be supported by the NC. This enables one to use the NC as if it were an X−Window terminal.

There are some HOWTOs for certain brands of NCs:

- JavaStation−HOWTO (by Sun)
- NC−HOWTO (IBM NetStation)
- NCD mini−HOWTO (NCD−ThinSTAR)
- NCD−X−Terminal mini−HOWTO
- XDM−and−X−Terminal mini−HOWTO

Hardware hookups

There are 3 different hardware arrangements for thin clients. One just uses a PC computer as a thin client by emulating a thin client. It really isn't a thin client but it behaves like one. Another type looks just like a text−terminal. It just looks like a monitor, with a connector for a keyboard and another connector for a network cable. It's a dedicated thin client and can't be used for anything else. The third type looks like a tiny computer. It uses a standard PC monitor and keyboard both of which plug into a small box which is a "thin" computer. This box provides an interface between the monitor/keyboard and the network.

History and the future

Promoters of NCs and related Window−Terminals projected that they would soon replace millions of PCs. In 1998 about 700,000 thin clients were sold with about 27% of them being NCs. But in 1999 only about 600,000 thin clients were sold. A major reason that more were not sold is that PCs have come down in price in recent years so that they are often no more expensive. However, it's argued that even though thin clients may cost the same as PCs, the maintenance and administration costs are less. Note that thin clients sometimes replace text terminals instead of PCs.

2.5 Emulation on a PC

Since a PC has a screen and keyboard (as does a terminal) but also has much more computing power, it's easy to use some of this computing power to make the PC computer behave like a text terminal. This is called "terminal emulation". They usually emulate text−terminals. See Terminal Emulation

---

3. Quick Install

This is a quick procedure to install a terminal without going through a Setup procedure for both the terminal and the host computer. It probably will not work right if the terminal happens to have been set up incompatible with the computer. If you don't understand some of it you'll need to consult other parts of this document for more info.

To install a terminal, first look in `/etc/termcap` or `terminfo.src` to find an entry for it (see Terminfo and Termcap (detailed)). Figure out what serial port you'll connect it to and what the tty designation is for that port (e.g. `ttyS1`, see Device Names). As the root user, edit `/etc/inittab` and add a getty command next to the other getty commands. The format of the getty command depends on which getty program you use. `agetty` (called just `getty` in the Debian distribution) is the easiest (no configuration file). See the "`info`" or "`man` re `getty`. For getty parameters use the terminfo (or termcap) name (such as vt100) for your terminal. Type in a baud−rate that the terminal supports. But if you set the baud too high you may need to use (See Flow Control).

Then physically connect the main serial port of the terminal to the chosen serial port of the computer with a null−modem cable and turn on the terminal. Don't expect most ready−made cables to be wired correctly for hardware flow control. Make sure the baud−rate of the terminal is set the same as you gave to getty and that its "data bits" is 8. Then at the computer console type "init q" to apply the changes you made to the inittab file. You should now see a login prompt at the terminal. If you don't, tap the terminal's return key. If this doesn't work read more of this document and/or see Trouble−Shooting.

4. Why Use a Terminal ?

4.1 Intro to Why Use a Terminal

PC's are so powerful today that just one PC can often support several persons using it at once, especially if they are doing low−load tasks such as text editing, data entry, etc. One way to do this is to connect a number of terminals to a single PC (or other host computer) by modems or direct cable connection. To do this, it's usually best to have a multi−user operating system such as Linux so that each user at a terminal can use the computer independently. This has been called "time sharing" but it's not good terminology today since "distributed" computing over a network is also a type of time sharing. It might be better described as "centralized" computing. But the central computer may be connected to the rest of the world via a network so that terminal users may send email, browse the Internet with the "lynx" browser, etc. So it's not exactly "centralized" either.

Terminals have seldom been used with PC's because the popular operating systems used for them (Windows, DOS, and Mac) were not multiuser until 1998 (available for MS Windows NT) and previously could not support terminals very well. Now that Linux, a multiuser operating system, is freely available for PC's, the use of terminals with PC's becomes more feasible. The drawback is that text terminals are not smart enough to support the type of graphical user interface (GUI) that many computer users today normally expect.

4.2 Lower Hardware Costs ?

When Computers (including PCs) were quite expensive, lower hardware costs was a significant advantage of using terminals. Today with cheap PCs, the cost savings is problematical. Here's what I wrote years ago when PCs were more expensive. It's still true today but of less significance.

If several people use the same computer as the same time, there is a reduction in the amount of hardware needed for the same level of service. One type of savings is due to code sharing. The application files on hard disks are shared as well as shared libraries in memory (even when people are running different programs provided they use some of the same functions in their code). Another type of savings is due to reduction of peak load. The hardware of a single PC may be idle most of the time as people slowly type in information, think, talk, or are away from their desks. Having several people on the same computer at once makes good use of much of this idle time which would otherwise be wasted.

These savings are substantial. One may roughly estimate (using statistical theory) that for 9 persons (8 terminals & 1 console) the shared PC only needs only about 3 times as much capacity (in memory, disk storage, CPU power, etc.) as a single PC in order to provide the same level of service per person. Thus the computational hardware for such a shared system should only cost about 1/3 as much per user. However, the cost of the display hardware (CRT's, keyboards, video electronics, etc.) is about the same for both cases. The terminals have the added cost of requiring additional serial ports at the host computer.

For a fair comparison with PC's, the terminals should have the same capabilities as the PC monitors. Unfortunately, color graphic terminals for Linux (X−windows) with high speed communication is a niche market with high prices so in this case there is not likely to be any savings in hardware costs. But for text terminals there will be some savings, especially if the terminals are obtained used at low cost.

4.3 Control of Software

For centralized computing, software (and the updates to software) only need be installed and configured on one host computer instead of several. The person in charge of this computer may control and configure the software which is installed on it. This is advantageous if the person controlling the host computer does an excellent job and knows about the needs and preferences of the other users. Users can be restricted in playing games or surfing the Internet by not installing the software (or by otherwise restricting access to it). Whether or not centralized control is desirable depends on the situation.

4.4 Hardware Upgrades

With terminals, the computer hardware upgrades take place on only one computer instead of many. This saves installation labor effort. While the cost of the hardware for the host computer upgrade will be more than that for a single PC (since the host needs more computing power than a PC), the cost will be significantly less than upgrading the hardware of a number of PC's being used instead of terminals.

4.5 Other Advantages of Terminals

- The elimination of noise from fans and disk drives provided the terminals are not close to the computer.
- The users of the terminals can share data and files and send e−mail to each other. It's similar to a local network.

4.6 Major Disadvantages of Terminals

- Text terminals have no high−speed graphic display (or high resolution graphics) although they can often use graphic character sets to draw boxes, etc. This lack limits the software that may be used on it.
- If the host computer goes down, then no one can use the terminals either (unless there is a "standby" host computer to connect to).

4.7 Are Text Terminals Obsolete ?

Text terminals are technologically obsolete because for a slightly higher cost of hardware, one could build a smarter terminal (with the same quality of display). This wasn't always the case since around 1980 memory cost thousands of dollars per megabyte. Today with low costs for memory and processors, one could turn a text terminal into a GUI graphic terminal for only about a 10% or 20% increase in hardware cost.

The reasons that text terminals are not fully obsolete are:

- The resolution of characters on the screen is better on monochrome terminals than for monitors in text mode.
- Many people don't need full screen graphics.
- Since running a text−terminal (in contrast to a GUI−graphics terminal) doesn't consume much of a modern PC's resources, a large number of terminals may be efficiently run from one PC.

---

5. Overview of How Terminals Work (in Linux)

See also section Some Details on How Terminals Work

5.1 Device Names

Each terminal is connected to a serial port on the host computer (often just a PC). The ports have names: ttyS0, ttyS1, ttyS2 etc. These are represented by special files found in the /dev (device) directory. /dev/ttyS0 corresponds to COM1 in DOS or Windows. ttyS1 is COM2, etc. See Terminal Special Files for details on these and related "devices" such as cua.

5.2 Login/Logout

When the host computer starts up it runs the program getty. The getty program runs the "login" program to log people in. See Getty (in /etc/inittab). A "login:" prompt appears on the screen. People at the terminals log in (after giving their passwords) and then have access to the computer. When it's time to shut the terminal down, one normally logs out and turns the terminal off. See Login Restrictions regarding restricting logins (including allowing the root user to log in at terminal).

5.3 Half/Full Duplex

If one watches someone typing at a terminal, the letters one types simultaneously appear on the screen. A naive person might think that what one types is being sent directly from the keyboard to the screen with a copy going to the computer (half−duplex like, see next paragraph). What is usually going on is that what is typed at the keyboard is directly sent only to the host computer which in turn echoes back to the terminal each character it receives (called full−duplex). In some cases (such as passwords or terse editor commands) the typed letters are not echoed back.

Full−duplex means that there are two (dual) one−way communication links. Full−duplex is the norm for terminals. Half−duplex is half of a duplex, meaning that there is only a single one−way communication link. This link must be shared by communications going in both directions and only one direction may be used at a time. In this case the computer would not be able to echo the characters you type (and send to it) so the terminal would need to also send each character you type directly to the terminal screen. Some terminals have a half−duplex mode of operation which is seldom used.

5.4 Terminal Memory

The image on a CRT tube will fade away almost instantly unless it is frequently redrawn on the screen by a beam of electrons shot onto the face of the tube. Since text sent to a terminal needs to stay on the screen, the image on the screen must be stored in the memory chips of the terminal and the electron beam must repeatedly scan the screen (say 60 times per second) to maintain the image. See Terminal Memory Details for more details.

5.5 Commands for the Terminal

The terminal is under the control of the computer. The computer not only sends the terminal text to display on the screen but also sends the terminal commands which are acted on. These are Control Codes (bytes) and escape sequences. For example, the CR (carriage return) control code moves the cursor the the left hand edge of the screen. A certain escape sequence (several bytes where the first byte is the "escape" control code) can move the cursor to the location on the screen specified by parameters placed inside the escape sequence.

The first terminals had only a few such commands but modern terminals have hundreds of them. The appearance of the display may be changed for certain regions: such as bright, dim, underline, blink, and reverse video. A speaker in a terminal can "click" when any key is pressed or beep if a mistake has occurred. Function keys may be programmed for special meanings. Various fonts may exist. The display may be scrolled up or down. Specified parts of the screen may be erased. Various types of flow control may be used to stop the flow of data when bytes are being sent to the terminal faster than the terminal can handle them. There are many more as you will see from looking over an advanced terminal manual or from the Internet links Esc Sequence List

5.6 Lack of Standardization Solved by Terminfo

While terminals made for the US all used the same ASCII code for the alphabet (except for IBM terminals which used EBCDIC), they unfortunately did not all use the same escape sequences. This happened even after various ANSI (and ISO) standards were established since these standards were never quite advanced enough. Furthermore, older terminals often lacked the capabilities of newer terminals. This might cause problems. For example, the computer might send a terminal an escape sequence telling it to split the screen up into two windows of specified size, not realizing that the terminal was incapable of doing this.

To overcome these problems a database called "termcap" (meaning "terminal capabilities") was established. Termcap was later superceded by "terminfo". This database resides in certain files on the computer and has a section of it (sometimes a separate file) for each model of terminal. For each model (such as VT100) a list of capabilities is provided including a list of certain escape sequences available. For example blink=\E5m means that to make the cursor start blinking the terminal must be sent: Escape 5 m. See Section Termcap and Terminfo (detailed) for more details. Application programs may utilize this database by calling certain C−Library functions. One large set of such programs (over 200) is named "ncurses" and are listed in the manual page for "ncurses".

5.7 The Interface

The environment variable TERM is the type of terminal Linux thinks you are using. Most application programs use this to look up the capabilities in the terminfo database so TERM needs to be set correctly. But there is more to a correct interface than the computer knowing about the capabilities of the terminal.

For bytes to flow from the computer to the terminal the terminal must be set to receive the bytes at the same baud rate (bits per second) as they are sent out from the terminal. If the terminal is set to receive at 19,200 baud and the computer sends out characters at 9600 baud, only garbage (or perhaps nothing) will be seen on the screen. One selects the baud rate for a terminal (as well as many other features) from the terminals "set−up" menus at the terminal. Most terminals have a large number of options in their "set−up" menus (see Terminal Set−Up (Configure) Details). The computer serial port has options also and these options must be set up in a compatible way (see Computer Set−Up (Configure) Details.

5.8 Emulation

Most terminals today have more than one emulation (personality or "terminal mode"). The terminal model numbers of terminals formerly made by DEC (Digital Equipment Corporation now Compaq) start with VT (e.g. VT100). Many other terminals which are not VT100 may be set up to emulate a VT100. Wyse is a major terminal manufacturer and most of their terminals can emulate various DEC terminals such at VT100 and VT220. Thus if you want to, say, use a VT320 terminal you may either use a real VT320 in "native" personality or possibly use some other terminal capable of emulating a VT320.

The "native" personalities usually have more capabilities so, other things being equal, "native" is usually the best to use. But other things may not be equal. Since the Linux console emulates a VT102 it you may want to have a terminal emulate this (or something close to it such as VT100). This will help insure that some programs that may not handle terminals properly will still work OK on your terminal. Some programs will assume that you are using a VT012 if the program can't find a terminfo for your terminal (or can't find a certain capability). Thus the failure of a program to work correctly with your non−vt102 terminal may well be your fault if you don't provide a good terminfo file for your terminal. Using "native" and then reporting any bugs will help discover and fix bugs which might not otherwise get detected.

The most common type of emulation is to use a PC like it was a vt100 terminal (or the like). Programs loaded into the PC's memory do the emulation. In Linux (unless you're in X−windows) the PC monitor (called the console) emulates a terminal of type "Linux" (close to vt100). Even certain windows within X−windows emulate terminals. See Terminal Emulation.

5.9 The Console

On a PC, the monitor is normally the console. It emulates a terminal of type "Linux". One logs on to it as a virtual terminal. See The Console: /dev/tty?. It receives messages from the kernel regarding booting and shutdown progress. One may have the messages that normally go to the console, go to the terminal. To get this you must manually patch the kernel, except that for kernel 2.2 (or higher) it is a "make config" option. See Make a Serial Terminal the Console.

---

6. Terminal Special Files such as /dev/tty

"tty" is an abbreviation for "Teletype". The first terminals were Teletypes (like remotely controlled typewriters). See subsection Teletypes. A list of Linux devices (the stuff in the /dev directory) may be found in "Linux Allocated Devices" which should be included with kernel sources. It "describes" what each device used for in only a word or two but doesn't tell you how to use them.

6.1 Serial Port Terminals

The computer considers each serial port to be a "device". It's sometimes called a terminal device since at one time terminals were the common use for the serial port. For each such serial port there is a special file in the /dev (device) directory. /dev/ttyS0 is the special file for the serial port known as COM1 in the DOS/Windows world. To send text to a terminal you may redirect standard output of some command−line command to the appropriate special file. For example typing "echo test > /dev/ttyS1" at the command prompt should send the word "test" to the terminal on ttyS1 (COM2) provided you have write permission on /dev/ttyS1. Similarly, typing "cat my_file > /dev/ttyS0" will send the contents of the file my_file to COM1 (ttyS0).

In addition to ttyS0 (/dev/ttyS0), ttyS1, ttyS2, etc. (the "S" stands for Serial port) there is also a "cua" series: cua0, cua1, cua2, etc. cua0 is the same port as ttyS0, etc. The "cu" of cua stands for CalloUt. The ttyS series are Posix compliant while using cua may permit the opening of a port that the modem control lines say is not ready. Starting with kernel version 2.2 cua is obsolete and a warning message is issued when you attempt to use it (although it still works). For the past few years it has only been included with Linux for backwards compatibility. A programmer can arrange things so that ttyS can behave just like cua, so cua is not really needed.

6.2 Pseudo Terminals

Pseudo terminals are pairs of devices such as /dev/ptyp3 and /dev/ttyp3. There is no physical device directly associated with either of them, not even a serial port connector. But if a program treats ttyp3 like it was a serial port, what is read and written to that port appears on the other member of the pair (ptyp3) which another program uses to read and write to. Thus two programs talk to each other via this method and one program (on ttyp3) thinks it's talking to a serial port.

You can't just take a pair of programs and make them talk to each other this way without modifying the source C code. Thus it's mainly programmers that must concern themselves with pseudo terminals and most users don't worry about them.

For example, if someone connects via telnet to your computer over a network, they may wind up connected to the device /dev/ptyp2 (a pseudo terminal port). The login process logs them in to /dev/ttyp2. Here the login program and the telnet program talk to each other via a "pseudo terminal". In X−Windows, the terminal emulator program, xterm (or rxvt), uses pseudo terminals. Ham radio programs under Linux also use them. Using certain application software it is possible to have 2 or more pseudo terminals attached to the same physical serial port.

For a pseudo terminal pair such as ptyp3 and ttyp3, the pty... is the master or controlling terminal and the tty... is the slave. There are only 16 ttyp's: ttyp0−ttypf (f is a hexadecimal digit). To get more pairs, the 3 letters q, r, s may be used instead of p. For example the pair ttys8, ptys8 is a pseudo terminal pair. The master and slave are really the same "port" but the slave is used by the application program and the master is used by a network program (or the like) which supplies (and gets) data to/from the slave port.

Unix98 doesn't use the above but instead uses a "pty master" which is /dev/ptmx. This can supply a pty on demand. While other unix−like systems have a manual page for pseudo terminals (may be named "pty") Linux lacks one. page devoted to only to pseudo terminals is needed for Linux. There is both a Linux pty module and a /usr/include/pty.h file.

6.3 The Controlling Terminal /dev/tty

/dev/tty stands for the controlling terminal (if any) for the current process. To find out which tty's are attached to which processes use the "ps −a" command at the shell prompt (command line). Look at the "tty" column. For the shell process you're in, /dev/tty is the terminal you are now using. Type "tty" at the shell prompt to see what it is (see manual pg. tty(1)). /dev/tty is something like a link to the actually terminal device name with some additional features for C−programmers: see the manual page tty(4).

6.4 /dev/ttyIN "Terminals"

N stands for an integer. One use of these in Linux is with the ISDN driver package: isdn4linux. The ttyIN is something like ttySN. There is also a cuiN which is something like cuaN. The ttyI and cui emulate modems and may be given modem commands.

6.5 The Console: /dev/ttyN

In Linux the PC monitor is usually called the console and has several device special files associated with it: tty0, tty1, tty2, etc. When you log in you are on tty1. To go to tty2 (on the same screen) For tty3 use Left Alt−F3, etc. These (tty1, tty2, tty3, etc.) are called "virtual terminals". tty0 is just an alias for the current virtual terminal and it's where messages from the system are sent. Thus messages from the system will be seen on the console (monitor) regardless of which virtual terminal it is displaying.

You may log in to different virtual terminals and thus have a few different sessions with the computer going on at the same time. Only the system or the root user may write to /dev/tty0 to which /dev/console is sometimes linked. For more info on the console see The Linux Console.

6.6 Creating a Device with "mknod"

The /dev directory comes supplied with many device special files. If you need something that's not there you may try to create it with the "mknod" command. See the manual page ttys(4) for how to do this for serial ports. To use `mknod` you must know the major and minor device numbers. You might be able to infer the numbers you need by using the "`ls -l`" command in the /dev directory. It will display the major and minor numbers of existing special files.

---

7. Some Details on How Terminals Work

If you know almost nothing about terminals, it's suggested that you first read Introduction and also read Overview of How Terminals Work.

7.1 Terminal Memory Details

The terminal screen refreshes itself at perhaps 60 times per second from an image stored in the memory of the terminal. For a PC the monitor's image is stored on the video card inside the computer but for a terminal, the equivalent of the video card is inside the terminal. For a text terminal the storage of the image uses little memory. Instead of putting every dot (pixel) on the screen into memory and requiring the storage of about a quarter−million dots, a much more efficient method of storage is used.

A screen−full of text may be represented inside the terminal memory by ASCII bytes, one for each character on the screen. An entire screen only takes about 2K ASCII bytes. To display these characters, the terminal must also know the bit−map (the shape) of each of the almost 100 printable ASCII characters. With a bit−map of a character using say 15 bytes, only about 1.5K of memory is needed for the bit−maps of all the ASCII characters (the font). This ASCII text and font memory is scanned so that the resulting image is put on the screen about 60 times each second. This is a form of shared memory where a single bit−map of a letter such as the letter e, is shared by all of the many letter e's which appear on a screen−full of text. Low memory requirements meant low costs to produce monitors in the early 1980's when the cost of memory was several thousand times higher than it is today (costing then several dollars per kilobyte).

7.2 Early Terminals

The first terminals were something like remotely controlled typewriters which could only "display" (print on paper) the character stream sent to them from the computer. The earliest models were called Teletypes. The name "tty" is just an abbreviation for "Teletype". Early terminals could do a line feed and a carriage return just like a typewriter and ring a bell when a bell character was received. Due to the lack of significant capabilities this was the first type of terminal to be labeled "dumb". This type of terminal interface (using a terminal type called "dumb") is sometimes used today when the computer can't figure out what kind of a terminal it is communicating with.

7.3 Escape Sequences and Control Codes (intro)

Terminals have many capabilities some of which are always present and some of which require commands from the computer to change or activate. To exercise all these capabilities under the control of the computer requires that special codes be established so that the computer can tell the terminal what to do. There are two major type of such codes: escape sequences and control codes (control characters). There are many times more escape sequences than control codes.

Control codes

The control codes (or control characters) consist of the first 32 bytes of the ASCII alphabet. They include the following: carriage−return (cursor to far left), line−feed (cursor down one line), backspace, escape−character, tab, and bell. They do not normally show on the screen. There is usually a command which you may give to your terminal which will result in them being displayed when they are received by the terminal. It's called something like "Display Controls" or "Monitor". If you do this then the display may look a mess since escape sequences, which all start with the ESC (escape) control character, are no longer executed. Words which should appear at the top or bottom of the screen will show up in other locations. The escape sequences to reposition the cursor display on the screen but the cursor doesn't move to where the escape sequence says.

Escape sequences

Since there are not nearly enough control codes to do everything (and for some reason, not all of them are utilized) many escape sequences are used. They consist of the "escape" (ESC) control character followed by a sequence of ordinary characters. Upon receiving an escape character, the terminal examines the characters following it so that it may interpret the sequence and carry out the intended command from the computer. Once it recognizes the end of a valid sequence, further characters received just display on the screen (unless they are control codes or more escape sequences). Some escape sequences may take parameters (or arguments) such as the coordinates on the screen to move the cursor to. The parameters become a part of the escape sequence. An Esc Sequence List is on the web for some terminals, but it's terse.

A list of the escape sequences for your terminal should be in the "programmers manual" for the terminal. Except for very old terminals, there may be two or three hundred such sequences. If you don't have a such manual it's not easy to find them. Some of the sequences are available on the Internet. One link is Esc Sequence List. By searching the Internet for one sequence (such as ESC[5m) you may come across a long list of them.

Another way to determine some of them is to find the terminfo entry (termcap) for the terminal and mentally decode it. See Terminfo and Termcap (detailed) in this document and/or the Termcap Manual on the Internet. Unfortunately, the terminfo (termcap) for a terminal often does not list all of the escape sequences which the terminal has available for use, but fortunately, the most important ones are usually there.

7.4 Display Attributes & Magic Cookies

Terminals have various methods of generating character attributes such as bold, reverse−video, underlining, etc. There should be no need for the user to worry about how how this is done, except that it creates problems for some old terminals and there is sometimes an option for this in the set−up menu of newer terminals.

The magic cookie method is obsolete. It's the simplest (and worst) method of defining attributes: Use a certain byte for the start of an attribute and another to end that attribute. For example, a "start underlining" magic cookie byte is placed just before the first word to be underlined. These extra bytes are put into the memory of the screen page, just like character bytes that display as characters. But this might foul up the count of the number of characters per line since non−printable magic cookie characters are intermingled with other printable characters. This sometimes causes problems.

A better method which uses more memory is to assign an attribute byte (or half=byte, etc.) to each displayed character. This method is used by PC video cards (for text) for the common PC monitor.

---

8. Special Features of Some Terminals

8.1 Color

While the common monochrome terminal is not a color terminal it may have a fixed "color" display other than white such as green or amber. All terminals have black (electron beam turned off = zero brightness). A real color terminal can change the color of the text and background to many different colors while a monochrome terminal can only change the brightness of a fixed color.

However, changing the brightness, etc. gives a lot of possibilities. For example, a black and white (monochrome) terminal can have white, grey, and black by varying the brightness. Some words can be black on a light grey background while other are highlighted by black on white. In addition there is white on black, underlining, and blinking.

Color works like the color on a computer monitor or TV screen. The CRT has three colors of dots on it with each color controlled by its own electron beam (3 beams). Monochrome has inherently better resolution since it doesn't depend on dots permanently fixed on the screen. For text terminals the only use of color is to differentiate text and this advantage is not always worth the cost of worse resolution. Thus monochrome may be better since it also costs less.

8.2 Multiple Sessions

For dual sessions the terminal has two serial ports of equal status. Each port is connected to a serial port on a different computer. Thus one may log in to two different computers with each session displaying in a split−screen window. Alternatively, each session may run full−screen with a "hot" key (or the like) to switch between sessions. One could also connect to two different serial ports on the same computer and log in twice (similar to "virtual terminals" at the console). The program "screen" will make any ordinary terminal (single session) connected to a single computer run two or more "sessions".

8.3 Printer/Auxiliary Port

Many terminals have a connector on the rear for such a port. It may be labeled as "Aux" or "Printer", etc. Some printer ports are for parallel printers while others are for serial printers. If a printer is connected to the printer or auxiliary port, then pressing certain keys will print the screen. One may also have everything that displays on the screen go also to the printer. If the port is an auxiliary port, one may connect this to another computer and almost have dual sessions as above. However, the video memory inside the terminal may not retain both sessions so you may need to refresh the screen when switching to the other session. There will likely not be a hot key either but possibly a programmable function key may be programmed to do this. There exists various key combinations and escape sequences for controlling such a port. See Printer Esc.

There is a program called `vtprint` which is designed to send a print job (text only) to your terminal to be printed on a printer attached to the terminal. It's homepage is `http://www.yavin.org/software/vtprint/` . It's also included (as of 1998) in the Debian distribution of Linux. `xprt` (also in Debian) seems to do something similar, but only for X−Window terminals ??

8.4 Pages

Many terminals permit the storage of more than one page in their video memory. Sometimes the page size is the same as the screen, but sometimes it is larger so that scrolling will reveal unseen parts of a page. So when one looks at a screen, there may be hidden text on the same page above or below the display. In addition, if there is more than just one page, there may be hidden text on these other pages. One use for pages is on terminals that support dual sessions. Each session may have its own page and one may switch back and forth between them.

Even if you only have a one−page−terminal with the page sized equal to what is displayed on the screen, you will still see other pages of a file (etc.) as the host sends more data to the terminal. One advantage to having additional pages stored in the terminal memory is so that you can jump to them instantly without waiting a second or so for them to be transmitted from the host.

Multiple pages is supported by ncurses. There is also a commercial program called "Multiscreen" which supports multiple pages but probably not for Linux ?? Multiscreen is reported to be part of SCO and is something like the virtual terminals on a Linux PC console. The Linux program "screen" makes it look like you have multiple pages but they are stored in the computer and but you can have only one page−like window for each running program.

8.5 Character−Sets

A character−set is normally represented by a list (or table or chart) of characters along with the byte code assigned to each character. The codes for a byte range from 0 to 255 (00 to FF in hexadecimal). In MS−DOS, character−set tables are called "code−pages". You should examine such a table if you're not familiar with them. They are sometimes included in printer and terminal manuals but also are found on the Internet.

ASCII is one of the most common character−sets used on text terminals. It is a 7−bit code but will usually work OK even if your terminal is set to 8−bit mode. In 8−bit mode with ASCII, the high order bit is always set to zero. Other character−sets are usually available and often use 8−bit codes (except on very old terminals where the only choice is ASCII). The first half of most character−sets are the conventional 128 ASCII characters and the second half (with the high−order bit set to 1) belong to a wide variety of character−sets. Character sets are often ISO standards. To get specialized character sets on a terminal, you may need to download a soft−font for that character−set into the memory of the terminal. Many terminals have a few built−in character sets (but perhaps not the one you need).

Here are some common 8−bit character sets. CP stands for Code Page character sets invented by IBM: CP−437 (DOS ECS), ISO−8859−1 (Latin−1), CP−850 (Multilingual Latin 1 −−not the same as ISO Latin−1), ANSI (derived from Latin−1). MS Windows uses ANSI while the Internet often uses Latin−1. There are several ISO−8859− character sets in addition to Latin−1. These include Greek (−7), Arabic (−6), Eastern European (−2), and a replacement for Latin−1 (−15) called Latin−9. There are many others. For example, KOI8−R is more commonly used for Russian than IS0−8859−5. Unicode is a very large character−set where each character is represented by 2 bytes instead on just one byte.

More info re character−sets are:

- Manual pages: ASCII and latin1
- HOWTO's for various languages (likely written in that language). See "Cyrillic" for Russian.
- A tutorial on character code issues Clearly written.
- Languages, Countries and Character Sets
- Languages of the World by Computers ...
- Links re Internationalization A long list of links (in Russian but most words in English).

Once you've found the character set name (or alpha−numeric designation) you are interested in, you may search for more info about it on the Internet.

8.6 Fonts

Most terminals made after the mid 1980's can accept downloaded soft−font. This means that they can display almost any character set provided that you can find the soft−font for it. If you can't find the needed soft−font, you can always create your own. A free font editor for this is called BitFontEdit (written by the author of this document) and (in 1998) was at
Europe: http://www.funet.fi/pub/culture/russian/comp/cyril−term/
N. America: http://metalab.unc.edu/pub/Linux/utils/terminal/ For mapping the keyboard (and screen) for use of various fonts see Character Mapping: mapchan

8.7 Keyboards & Special Keys

Terminal keyboards often have a number of keys that one doesn't find on a PC keyboard. Few (if any) actual terminals will have all of these keys and most will have additional keys not listed here. Some have a large number of special purpose keys such as terminals made for use with cash registers. There are often many more key meanings than shown here since these keys often have extended meanings when used in conjunction with other keys (such as shift and control).

- BREAK sends a very long 0 bit (space = +12 V) of duration 300 to 700 milliseconds to the host. The host may interpret this as an interrupt if stty has set brkint or ignore it if ignbrk is set.
- NO SCROLL stops the screen from scrolling like ^S does. Depressing it again resumes scrolling. Uses flow control signals to do this.
- REPEAT if held down with an other key, forces repeated output of that other key even if the auto−repeat option is set to off.

- LINE FEED sends the line feed character ^J to the host. Seldom used.
- SET−UP allows the manual configuration of the terminal via menus. Sometimes purposely disabled by putting a block under it so it can't be pressed down. Sometimes another key such as shift or control must be pressed at the same time. See Getting Into Set−Up (Configuration) Mode.
- LOCAL disconnects the terminal from the host. In local, what one types goes directly to the screen. Useful for testing.
- RETURN is the same as the "enter" key on a PC. It usually sends a carriage return to the host which normally get translated to a new−line character by the host's device driver. On some terminals it may be set up to send something else.
- F1, F2, ... or PF1, PF2, ... are function keys which usually may be programmed to send out a sequence of bytes (characters). See Function Keys

9. Terminal Emulation; the Console

9.1 Intro to Terminal Emulation

Since a PC has a screen and keyboard (as does a terminal) but also has much more computing power, it's easy to use some of this computing power to make the PC computer behave like a text terminal. This is one type of terminal emulation. Another type of terminal emulation is where you set up a real terminal to emulate another brand/model of terminal. To do this you select the emulation you want (called "personality" in Wyse jargon) from the terminal's set−up menu. This section is about the first type of emulation: emulating a terminal on a PC.

Emulation software is available for MS Windows and comes built−in with recent versions of MS Windows. Most Linux free software can only emulate a VT100, VT102, or VT100/ANSI. If you find out about any others, let me know. Since most PC's have color monitors but VT100 and VT102 were designed for a monochrome monitor, the emulation usually adds color capabilities (and a choice of colors). Sometimes the emulation is not 100% perfect but this usually causes few problems. For using a Mac computer to emulate a terminal see the mini−howto: Mac−Terminal.

9.2 Don't Use TERM For Emulation

Some have erroneously thought that they could create an emulator at a Linux console (monitor) by setting the environment variable TERM to the type of terminal they would like to emulate. This does not work. The value of TERM only tells an application program what terminal you are using. This way it doesn't need to interactively ask you this question. If you're at the PC monitor it's a terminal of type "Linux" and your can't change this. So you must set TERM to "Linux".

If you set it to something else you are fibbing to application programs. As a result they will incorrectly interpret certain escape sequences from the console resulting in a corrupted interface. Since the Linux console behaves almost like a vt100 terminal, it could still work almost OK if you falsely claimed it was a vt100 (or some other terminal which is something like a vt100). It may seeming work OK most of the time but once in a while will make a mistake when editing or the like.

9.3 Communication (Dialing) programs

Dialing programs for making a PPP connection to the Internet don't normally include any terminal emulation. But some other modem dialing programs (such as minicom or seyon) do. Using them one may (for example) dial up public libraries to use their catalogs and indexes, (or even read magazine articles). They are also useful for testing modems. Seyon is only for use with X−windows and can emulate Tektronix 4014 terminals.

The communication program kermit doesn't do terminal emulation as it is merely a semi−transparent pipe between whatever terminal you are on and the remote site you are connected to. Thus if you use kermit on a Linux PC the terminal type will be "Linux". If you have a Wyse60 connected to your PC and run kermit on that, you will appear as a Wyse60 to the remote computer (which may not be able to handle Wyse60 terminals). Minicom emulates a VT102 and if you use it on Wyse60 terminal the Wyse escape sequences will get translated to VT102 escape sequences before the data goes out to the modem. Kermit can't do this.

Emulators exist under DOS such as `telix` and `procomm` work just as well. The terminal emulated is often the old VT100, VT102, or ANSI (like VT100).

Emulation under X−Windows

Xterm (obsolete ??) may be run under X−Windows which can emulate a VT102, VT220, or Tektronix 4014. There is also an xterm emulation (although there is no physical terminal named "xterm"). If you don't need the Tektronix 4014 emulation (a vector graphics terminal; see Graphics Terminals) you may use `eterm`. Predecessors to `eterm` are `rxvt` and `xvt`. `eterm` supports pixmaps.

For non−Latin alphabets, kterm is for Kanji terminal emulation (or for other non−Latin alphabets) while xcin is for Chinese. There is also 9term emulation. This seems to be more than just an emulator as it has a built−in editor and scroll−bars. It was designed for Plan 9, a Unix−like operating system from AT&T.

Real terminals better

Unless you are using X−Windows with a large display, a real terminal is often nicer to use than emulating one. It usually costs less, has better resolution for text, and has no disk drives to make annoying noises.

9.4 Testing Terminal Emulation

For the VT series terminals there is a test program: `vttest` to help determine if a terminal behaves correctly like a vt53, vt100, vt102, vt220, vt320, vt420 etc. There is no documentation but it has menus and is easy to use. To compile it run the configure script and then type "make". It may be downloaded from: http://sunsite.unc.edu/pub/Linux/utils/console/

9.5 The Linux Console

The console for a PC Linux system is normally the computer monitor. It emulates a terminal of type "Linux". There is no way (unless you want to spend weeks rewriting the kernel code) to get it to emulate anything else. Setting the TERM environment variable to type of terminal other than "Linux" will not result in emulating that other terminal. It will only result in a corrupted interface since you have falsely declared (via the TERM variable) that your "terminal" is of a type different from what it is. See Don't Use TERM For Emulation

The "Linux" emulation is flexible and has features which go well beyond those of the vt102 terminal which it was intended to emulate. These include the ability to use custom fonts and easily re−map the keyboard (without patching the source code and recompiling the kernel as is required for the case of a real terminal). These extra features reside in the console driver software and not in the emulation software but the results are like it was part of the emulation.

Many commands exist (see Keyboard−and−Console−HOWTO) to utilize these added features. Real terminals, which use neither scan codes nor VGA cards, unfortunately can't use most of these features. One may recompile Linux to make a terminal receive the messages which normally go to the console. See Make a Serial Terminal the Console.

9.6 Emulation Software

Emulators often don't work quite right so before purchasing software you should try to throughly check out what you will get.

Make a Linux PC a terminal

Unless you want to emulate the standard vt100 (or close to it). There doesn't seem to be much free terminal emulation software available for Linux. The free programs minicom and seyon (only for X−windows) can emulate a vt100 (or close to it). Seyon can also emulate a Tektronix 4014 terminal.

There's a specialized Linux distribution: Serial Terminal Linux. It will turn a PC to into a minicom−like terminal. It's small (fits on a floppy) and will not let you use the PC for any other purpose (when it's running). See http://members.wri.com/johnnyb/seriallinux/. It will let you have more than one session running (similar to virtual terminals), one for each serial port you have.

TERM (non−free from Century Software) http://www.ecc400.com/censoft/termunix.html can emulate Wyse60, 50; VT 220, 102, 100, 52: TV950, 925, 912; PCTERM; ANSI; IBM3101; ADM−1l; WANG 2110. Block mode is available for IBM and Wyse. It runs on a Linux PC.

Make a non−Linux PC a terminal

Emulators exist which run on non−Linux PCs. They permit you to use a non−Linux−PC as a terminal connected to a Linux−PC. Under DOS there is `telix` and `procomm`. Windows comes with "HyperTerminal" (formerly simply called "Terminal" in Windows 3.x and DOS). Competing with this is "HyperTerminal Private Edition" http://www.hilgraeve.com/htpe/index.html which is non−free to business. It can emulate vt−220. Turbosoft's (Australia) TTWin http://www.turbosoft.com.au/ can emulate over 80 different terminals under Windows.

For the Mac Computer there is emulation by Carnation Software http://www.carnationsoftware.com/carnation/HT.Carn.Home.html

One place to check terminal emulation products is Shuford's site, but it seems to lists old products (which may still work OK). The fact that most only run under DOS (and not Windows) indicates that this info is dated. See http://www.cs.utk.edu/~shuford/terminal/term_emulator_products.txt.

---

10. Flow Control (Handshaking)

Flow control (= handshaking = pacing) is to prevent too fast of a flow of bytes from overrunning a terminal, computer, modem or other device. Overrunning is when a device can't process what it is receiving quickly enough and thus loses bytes and/or makes other serious errors. What flow control does is to halt the flow of bytes until the terminal (for example) is ready for some more bytes. Flow control sends its signal to halt the flow in a direction opposite to the flow of bytes it wants to stop. Flow control must both be set at the terminal and at the computer.

There are 2 types of flow control: hardware and software (Xon/Xoff or DC1/DC3). Hardware flow control uses dedicated signal wires such as RTS/CTS or DTR/DSR while software flow control signals by sending DC1 or DC3 control bytes in the normal data wires. For hardware flow control, the cable must be correctly wired.

The flow of data bytes in the cable between 2 serial ports is bi−directional so there are 2 different flows (and wires) to consider:

1. Byte flow from the computer to the terminal
2. Byte flow from the terminal keyboard to the computer.

10.1 Why Is Flow Control Needed ?

You might ask: "Why not send at a speed slow enough so that the device will not be overrun and then flow control is not needed?" This is possible but it's usually significantly slower than sending faster and using flow control. One reason for this is that one can't just set the serial port baud rate at any desired speed such as 14,500, since only a discrete number of choices are available. The best choice is to select a rate that is a little higher than the device can keep up with but then use flow control to make things work right.

If one decides to not use flow control, then the speed must be set low enough to cope with the worst case situation. For a terminal, this is when one sends escape sequences to it to do complex tasks that take more time than normal. In the case of a modem (with data compression but no flow control) the speed from the computer to the modem must be slow enough so that this same speed is usable on the phone line, since in the worst case the data is random and can't be compressed. If one failed to use flow control, the speed (with data compression turned on) would be no faster than without using any compression at all.

Buffers are of some help in handling worst case situations of short duration. The buffer stores bytes that come in too fast to be processed at once, and saves them for processing later.

10.2 Padding

Another way to handle a "worst case" situation (without using flow control or buffers) is to add a bunch of nulls (bytes of value zero) to escape sequences. Sometimes DEL's are used instead provided they have no other function. See Recognize Del.

The escape sequence starts the terminal doing something, and while the terminal is busy doing it, it receives a bunch of nulls which it ignores. When it gets the last null, it has completed its task and is ready for the next command. This is called null padding. These nulls formerly were called "fill characters". These nulls are added just to "waste" time, but it's not all wasted since the terminal is usually kept busy doing something else while the nulls are being received. It was much used in the past before flow control became popular. To be efficient, just the right amount of nulls should be added and figuring out this is tedious. It was often done by trial and error since terminal manuals are of little or no help. If flow control doesn't work right or is not implemented, padding is one solution. Some of the options to the `stty` command involve padding.

10.3 Overrunning a Serial Port

One might wonder how overrunning is possible at a serial port since both the sending and receiving serial ports involved in a transmission of data bytes are set for the same speed (in bits/sec) such as 19,200. The reason is that although the receiving serial port electronics can handle the incoming flow rate, the hardware/software that fetches and processes the bytes from the serial port sometimes can't cope with the high flow rate.

One cause of this is that the serial port's hardware buffer is quite small. Older serial ports had a hardware buffer size of only one byte (inside the UART chip). If that one received byte of data in the buffer is not removed (fetched) by CPU instructions before the next byte arrives, that byte is lost (the buffer is overrun). Newer UART's, namely most 16550's, have 16−byte buffers (but may be set to emulate a one−byte buffer) and are less likely to overrun. It may be set to issue an interrupt when the number of bytes in its buffer reaches 1, 4, 8, or 14 bytes. It's the job of another computer chip (usually the main CPU chip for a computer) to take these incoming bytes out of this small hardware buffer and process them (as well as perform other tasks).

When contents of this small hardware receive buffer reaches the specified limit (one byte for old UART'S) an interrupt is issued. Then the computer interrupts what it was doing and software checks to find out what happened. It finally determines that it needs to fetch a byte (or more) from the serial port's buffer. It takes these byte(s) and puts them into a larger buffer (also a serial port buffer) that the kernel maintains in main memory. For the transmit buffer, the serial hardware issues an interrupt when the buffer is empty (or nearly so) to tell the CPU to put some more bytes into it to send out.

Terminals also have serial ports and buffers similar to the computer. Since the flow rate of bytes to the terminal is usually much greater than the flow in the reverse direction from the keyboard to the host computer, it's the terminal that is most likely to suffer overrunning. Of course, if you're using a computer as a terminal (by emulation), then it is likewise subject to overrunning.

Risky situations where overrunning is more likely are: 1. When another process has disabled interrupts (for a computer). 2. When the serial port buffer in main (or terminal) memory is about to overflow.

10.4 Stop Sending

When its appears that the receiver is about to be overwhelmed by incoming bytes, it sends a signal to the sender to stop sending. That is flow control and the flow control signals are always sent in a direction opposite to the flow of data which they control (although not in the same channel or wire). This signal may either be a control character (^S = DC3 = Xoff) sent as an ordinary data byte on the data wire (in−band signalling), or a voltage transition from positive to negative in the dtr−to−cts (or other) signal wire (out−of−band signalling). Using Xoff is called "software flow control" and using the voltage transition in a dedicated signal wire (inside the cable) is called hardware flow control.

10.5 Keyboard Lock

With terminals, the most common case of "stop sending" is where the terminal can't keep up with the characters being sent to it and it issues a "stop" to the PC. Another case of this is where someone presses control−S. Much less common is the opposite case where the PC can't keep up with your typing speed and tells the terminal to stop sending. The terminal "locks" its keyboard and a message or light should inform you of this. Anything you type at a locked keyboard is ignored.

The term "locked" is also sometimes used for the common case of where the computer is told to stop sending to a terminal. The keyboard is not locked so that whatever you type goes to the computer. Since the computer can't send anything back to you, characters you type don't display on the screen and it may seem like the keyboard is locked. Scrolling is locked (scroll lock) but the keyboard is not locked.

10.6 Resume Sending

When the receiver has caught up with its processing and is ready to receive more data bytes it signals the sender. For software flow control this signal is the control character ^Q = DC1 = Xon which is sent on the regular data line. For hardware flow control the voltage in a signal line goes from negative (negated) to positive (asserted). If a terminal is told to resume sending the keyboard is then unlocked and ready to use.

10.7 Hardware Flow Control (RTS/CTS etc.)

Some older terminals have no hardware flow control while others used a wide assortment of different pins on the serial port for this. For a list of various pins and their names see Standard Null Modem Cable Pin−out. The most popular pin to use seems to be the DTR pin (or both the DTR pin and the DSR pin).

RTS/CTS, DTR, and DTR/DSR Flow Control

Linux PC's use RTS/CTS flow control, but DTR/DSR flow control (used by some terminals) behaves similarly. DTR flow control (in one direction only and also used by some terminals) is only the DTR part of DTR/DSR flow control.

RTS/CTS uses the pins RTS and CTS on the serial (EIA−232) connector. RTS means "Request To Send". When this pin stays asserted (positive voltage) at the receiver it means: keep sending data to me. If RTS is negated (voltage goes negative) it negates "Request To Send" which means: request not to send to me (stop sending). When the receiver is ready for more input, it asserts RTS requesting the other side to resume sending. For computers and terminals (both DTE type equipment) the RTS pin sends the flow control signal to the CTS pin (Clear To Send) on the other end of the cable. That is, the RTS pin on one end of the cable is connected to the CTS pin at the other end.

For a modem (DCE equipment) it's a different scheme since the modem's RTS pin receives the signal and its CTS pin sends. While this may seem confusing, there are valid historical reasons for this which are too involved to discuss here.

Terminals usually have either DTR or DTR/DSR flow control. DTR flow control is the same as DTR/DSR flow control but it's only one−way and the DSR pin is not used. For DTR/DSR flow control at a terminal, the DTR signal is like the signal sent from the RTS pin and the DSR pin is just like the CTS pin.

Connecting up DTR or DTR/DSR Flow Control

Some terminals use only DTR flow control. This is only one−way flow control to keep the terminal from being overrun. It doesn't protect the computer from someone typing too fast for the computer to handle it. In a standard null modem (crossover) cable the DTR pin at the terminal is connected to the DSR pin at the computer. But Linux doesn't support DTR/DSR flow control (although drivers for some multiport boards may support DTR/DSR flow control.) A way around this problem is to simply wire the DTR pin at the terminal to connect to the CTS pin at the computer and set RTS/CTS flow control (stty crtscts). The fact that it's only one way will not affect anything so long as the host doesn't get overwhelmed by your typing speed and drop RTS in a vain attempt to lock your keyboard. See Keyboard Lock. For DTR/DSR flow control (if your terminal supports this two−way flow control) you do the above. But you also connect the DSR pin at the terminal to the RTS pin at the computer. Then you are protected if you type too fast.

Old RTS/CTS handshaking is different

What is confusing is that there is the original use of RTS where it means about the opposite of the previous explanation above. This original meaning is: I Request To Send to you. This request was intended to be sent from a terminal (or computer) to a modem which, if it decided to grant the request, would send back an asserted CTS from its CTS pin to the CTS pin of the computer: You are Cleared To Send to me. Note that in contrast to the modern RTS/CTS bi−directional flow control, this only protects the flow in one direction: from the computer (or terminal) to the modem.

For older terminals, RTS may have this meaning and goes high when the terminal has data to send out. The above use is a form of flow control since if the modem wants the computer to stop sending it drops CTS (connected to CTS at the computer) and the computer stops sending.

Reverse Channel

Old hard−copy terminals may have a reverse channel pin (such as pin 19) which behaves like the RTS pin in RTS/CTS flow control. This pin but will also be negated if paper or ribbon runs out. It's often feasible to connect this pin to the CTS pin of the host computer. There may be a dip switch to set the polarity of this signal.

10.8 Is Hardware Flow Control Done by Hardware ?

Some think that hardware flow control is done by hardware but only a small part of it is done by hardware. Most of it is actually done by your operating system software. UART chips and associated hardware usually know nothing at all about hardware flow control. When a hardware flow control signal is received (due to the signal wire flipping polarity) the hardware gives an electrical interrupt signal to the CPU. However, the hardware has no idea what this interrupt means. The CPU stops what it was doing and jumps to a table in main memory that tells the CPU where to go to find a program which will find out what happened and determine what to do about it. In this case this program stops the outgoing flow of bytes.

But even before this program stops the flow, it was already stopped by the interrupt which interrupted the work of the CPU. This is one reason why hardware flow control stops the flow faster. It doesn't need to wait for a program to do it. But if that program didn't command that the flow be stopped, the flow would resume once that program exited. So the program is essential to stop the flow even though it is not the first to actually stop the flow. After the interrupt happens any bytes (up to 16) which were already in the serial port's hardware transmit buffer will still get transmitted. So even with hardware flow control the flow doesn't instantly stop.

Using software flow control requires that each incoming byte be checked to see if it's an "off" byte. These bytes are delayed by passing thru the 16−byte receive buffer. If the "off" byte was the first byte into this buffer, there could be a wait while 15 more bytes were received. Then the 16 bytes would get read and the "off" byte found. This extra delay doesn't happen with hardware flow control.

10.9 Obsolete ?? ETX/ACK or ENQ/ACK Flow Control

This is also software flow control and requires a device driver that knows about it. Bytes are sent in packets (via the async serial port) with each packet terminated by an ETX (End of Text) control character. When the terminal gets an ETX it waits till it is ready to receive the next packet and then returns an ACK (Acknowledge). When the computer gets the ACK, it then send the next packet. And so on. This is not supported by Linux ?? Some HP terminals use the same scheme but use ENQ instead of ETX.

11. Physical Connection

Multiport boards allow many terminals (or modems) to be connected to one PC computer. A terminal may be connected to its host computer either by a direct cable connection, via a modem, or via a terminal server.

11.1 Multiport I/O Cards (Adapters)

Additional serial cards may be purchased which have many serial ports on them called "multiport boards". These boards are not covered in this HOWTO but there is a list of them (with URLs) in the Serial−HOWTO.

11.2 Direct Cable Connection.

The simplest way to connect a terminal to a host computer is via a direct connection to a serial port on the computer. You may also use some the info in this section for connecting one computer to another (via the serial port). Most PC's come with a couple of serial ports, but one is usually used by a mouse. For the EIA−232 port, you need a null modem cable that crosses over the transmit and receive wires. In ethernet terminology it would be called a "crossover cable" (but the ethernet cable will not work for the serial port). If you want hardware flow control, you will probably use the DTR pin (or both the DTR and DSR pins).

Make sure you have the right kind of cable. A null modem cable bought at a computer store may do it (if it's long enough), but it probably will not work for hardware flow control. Such a cable may be labeled as a serial printer cable. Only larger computer stores are likely to stock such cables. A "modem cable" will not work since the wires go straight thru (and don't cross over). See Buy or Make your own cable. Make sure you are connecting to your PC's serial port at the male DB25 or the DB9, and not to your parallel port (female DB25).

Null Modem cable pin−out (3, 4, or 5 conductor)

These 3 diagrams are for real text−terminals. But you could use them to connect up 2 PCs if you substitute RTS for DTR and CTS for DSR. (Don't use 4−conductors for PC−to−PC). For terminals, if you only have DTR flow control (one−way) you may eliminate the RTS−to−DSR wire. If you have no hardware flow control, then you may also eliminate the CTS−to−DTR wire. Then if you have 2@ twisted pairs, you may then use 2 wires for signal ground per A Kludge using Twisted−Pair Cable. For a DB25 connector on your PC, you need:

```
PC male DB25                                Terminal DB25
  TxD   Transmit Data       2 --> 3             RxD   Receive Data
  RxD   Receive Data        3 <-- 2             TxD   Transmit Data
  SG    Signal Ground       7 --- 7             SG    Signal Ground
  CTS   Clear To Send       5 <--20             DTR   Data Terminal Ready
  RTS   Request To Send     4 --> 6             DSR   Data Set Ready
```

If you have a DB9 connector on your PC, try the following:

```
      PC DB9                                       Terminal DB25
```

```
RxD   Receive Data        2 <-- 2            TxD   Transmit Data
TxD   Transmit Data       3 --> 3       RxD   Receive Data
SG    Signal Ground       5 --- 7       SG    Signal Ground
CTS   Clear To Send       8 <--20       DTR   Data Terminal Ready
RTS   Request To Send     7 --> 6       DSR   Data Set Ready   **
```

If you have a DB9 connector on both your serial port and terminal:

```
    PC DB9                                     Terminal DB9
RxD   Receive Data        2 <-- 3            TxD   Transmit Data
TxD   Transmit Data       3 --> 2       RxD   Receive Data
SG    Signal Ground       5 --- 5       SG    Signal Ground
CTS   Clear To Send       8 <-- 4       DTR   Data Terminal Ready
RTS   Request To Send     7 --> 6       DSR   Data Set Ready   **
```

The above don't have modem control lines so be sure to give a "local" option to getty (which is equivalent to "stty clocal"). Also if you need hardware flow control it must be enabled at your computer (use a −h flag with agetty) (equivalent to "stty crtscts").

Standard Null Modem cable pin−out (7 conductor)

The following 3 diagrams show full "standard" null modem cables. One that you purchase may be wired this way. Another pinout is for 20 and 6 to cross over and to have 8 cross over to both 4 and 5. This will not provide hardware flow control (RTS/CTS) for directly connected computers. Both of the above will work for terminals using software (Xon/Xoff) flow control (or no flow control). None of these cables will work for terminal hardware flow control since most real terminals support DTR or DTR/DSR flow control (handshaking) but Linux doesn't yet (2000).

```
PC male DB25                                Terminal DB25
TxD   Transmit Data         2 --> 3     RxD   Receive Data
RxD   Receive Data          3 <-- 2     TxD   Transmit Data
RTS   Request To Send       4 --> 5     CTS   Clear To Send
CTS   Clear To Send         5 <-- 4     RTS   Request To Send
DSR   Data Set Ready        6
                            |
DCD   Carrier Detect        8 <-- 20    DTR   Data Terminal Ready
SG    Signal Ground         7 --- 7     SG    Signal Ground
                                  6     DSR   Data Set Ready
                                  |
DTR   Data Terminal Ready  20 --> 8     DCD   Carrier Detect
```

Alternatively, a full DB9−DB25 null modem cable (will not work with terminal hardware handshaking; see above):

```
PC DB9                                      Terminal DB25
RxD   Receive Data          2 <-- 2     TxD   Transmit Data
TxD   Transmit Data         3 --> 3     RxD   Receive Data
                                  6     DSR   Data Set Ready
                                  |
DTR   Data Terminal Ready   4 --> 8     DCD   Carrier Detect
SG    Signal Ground         5 --- 7     SG    Signal Ground
DCD   Carrier Detect        1
                            |
DSR   Data Set Ready        6 <-- 20    DTR   Data Terminal Ready
RTS   Request To Send       7 --> 5     CTS   Clear To Send
CTS   Clear To Send         8 <-- 4     RTS   Request To Send
(RI   Ring Indicator        9 not needed)
```

(Yes, the pins 2 and 3 really do have opposite meanings for DB9 and DB25 connectors!)

Here's how to null−modem connect two DB9's together (but DTR flow control will not work):

```
PC DB9                                  DB9
RxD   Receive Data          2 <-- 3     TxD   Transmit Data
TxD   Transmit Data         3 --> 2     RxD   Receive Data
                                  6     DSR   Data Set Ready
                                  |
DTR   Data Terminal Ready   4 --> 1     DCD   Carrier Detect
GND   Signal Ground         5 --- 5     GND   Signal Ground
DCD   Carrier Detect        1
                            |
DSR   Data Set Ready        6 <-- 4     DTR   Data Terminal Ready
RTS   Request To Send       7 --> 8     CTS   Clear To Send
CTS   Clear To Send         8 <-- 7     RTS   Request To Send
RI   Ring Indicator         9 (not used)
```

Using the above 2 connections provide full modem control signals and seemingly allow one to set "stty −clocal". Then one must turn on the terminal first (asserts DTR) before the port may be opened in a normal manner by getty, etc. But there is likely to be trouble if you fail to turn on the terminal first (see Getty Respawning Too Rapidly). For this reason one should use "stty clocal" which is the default (ignores modem control lines) and the additional wires in these cables then serve no useful purpose.

In olden days when it may not have been this easy to ignore modem control signals etc, the following "trick" was done for cables that lacked conductors for modem control: on your computer side of the connector, connect RTS and CTS together, and also connect DSR, DCD and DTR together. This way, when the computer needs a certain handshaking signal to proceed, it will get it (falsely) from itself.

Overcoming length limitations

A cable longer than a 50 feet or so may not work properly at high speed. Much longer lengths sometimes work OK, especially if the speed is low and/or the cable is a special low−capacitance type and/or the electronics of the receiving end are extra sensitive. It is claimed that under ideal conditions at 9600 baud, 1000 feet works OK. One way to cover long distances is to install 2@ line drivers near each serial port so as to convert unbalanced to balanced (and conversely) and then use twisted pair cabling. But line drivers are expensive.

Another way to increase the distance is to try to cancel out much of the magnetic field created by the currents in the transmit and receive data wires: TxD and RxD. To do this, ground return lines, which have current which is roughly equal (but in the opposite direction) are placed next to the the transmit and received wires. Twisted pair has the best cancellation. Some DEC terminals have two signal ground wires for this purpose. For example, one pair would be TxD and SG(TxD) where SG is signal ground. If you use ribbon cable, insure that the TxD and SG(TxD) wires are right next to each other. Similarly for the RxD.

If there is only one signal ground wire provided by both the PC and the terminal, it may be split into two wires in a twisted pair cable for this purpose. You might think that return currents will be equally split between the two signal ground wires. This would cancel out only about half of the magnetic field. But it's better cancellation than this because return current prefers the path of least impedance. The return path of a data signal (such as TxD) has the lowest impedance (due to lower inductance) if it flows back in the same twisted pair. Although I've haven't seen any experimental test results for this method, it should allow longer cable lengths.

Hardware Flow Control cables

If you expect to use hardware flow control (handshaking) you will likely need to make up your own cable (or order one made). Of course, if the connecters on the ends of a used cable remove, you might rewire it. See Installing DB Connectors. You will need to determine whether or not the terminal uses the DTR pin for this, and if not, what pin (or pins) it uses. The set−up menus may give you a clue on this since there may be an option for enabling "DTR handshaking" (or flow control) which of course implies that it uses the DTR pin. It may also use the DSR pin. See Hardware Flow Control for a detailed explanation of it. Older terminals may have no provision for hardware flow control.

Cable tips

The normal "straight thru" cable will not work unless you are using it as an extension cable in conjunction with either a null modem (crossover) cable or a null modem adapter. Make sure that the connectors on the cable ends will mate with the connectors on the hardware. One may use telephone cable which is at least 4−conductor (and possibly twisted pair). Shielded, special low−capacitance cable computer cable is best.

A kludge using twisted−pair cable

See also Overcoming Length Limitations. Although none of the EIA−232 signals are balanced for twisted pair one may attempt to use twisted−pair cable with it. Use one pair for transmit and another for receive. To do this connect signal ground to one wire in each of these 2 pair. Only part of the signal ground current flows in the desired wire but it may help. Due to the lower inductance of the twisted pair circuit (as compared to ground return current by some other path) more return (ground) current will confine itself to the desired twisted pair than one would expect from only resistance calculations. This is especially true at higher frequencies since inductive impedance increases with frequency. The rectangular wave of the serial port contains high frequency harmonics.

Cable grounding

Pin 1 (of a DB25) should be chassis ground (also earth ground) but on cheap serial ports it may not even be connected to anything. A 9−pin connector doesn't even have a chassis ground. The signal ground is pin 7 and is usually grounded to chassis ground. This means that part of the signal current will flow thru the ground wires of the building wiring (undesirable). Cable shields are supposed to be only grounded at one end of the cable, but it may be better to ground both ends since it's better to have current in the shield than in the building wiring ??

11.3 Modem Connection

Using a terminal−modem combination (without a computer) one may connect to BBS's. Some BBS's (such a free−nets) permit Internet access via the text browser lynx which will work on text terminals. Thus with an old terminal and external modem, one may connect to the Internet. If one connects to a host computer on which one has an account, then one may sometimes store ones work (or downloads) there.

Dialing out from a terminal

Instead of connecting a terminal (or computer emulating a terminal) directly to a host computer using a cable it may be connected to the host via a telephone line (or dedicated private line) with a modem at each end of the line. The terminal (or computer) will usually dial out on a phone line to a host computer.

Most people use a PC and modem for dialing out. The PC could have a terminal connected to a serial port and the person at the terminal may dial out using the PC. Connecting a real terminal directly to an external modem is more difficult since the real terminal isn't very intelligent and doesn't give as much feedback to the user. For dialing out, many terminals can store one or more telephone numbers as messages which may be "set−up" into them and are sent out to the modem by pressing certain function keys. Many modems can also store phone numbers. The modem initiation sequence must precede the telephone number. When the outgoing call is answered by another modem at the other end of the phone line, the the host computer on this modem may run a getty program to enable you to log in.

Terminal gets dialed into

It's common for a computer running Linux to get dialed into. The caller gets a login prompt and logs in. At first glance, it may seem strange how a dumb terminal (not connected to any computer) could accept an incoming call, but it can. One possible reason for doing this is to save on phone bills where rates are not symmetric. Your terminal needs to be set up for dial−in: Set the modem at your terminal for automatic answer (Register S0 set to 2 will answer on the 2nd ring). You turn on the terminal and modem before you expect a call and when the call comes in you get a login prompt and log in.

The host computer that dials out to your terminal needs to do something quite unusual. As soon as your modem answers, it needs to run login (getty). A host may do this by running the Linux program "callback" sometimes named "cb". Callback is for having computer A call computer B, and then B hangs up and calls A back. This is what you want if you are using computer A to emulate a terminal. For the case of a real terminal this may be too complex a task so the host may utilize only the "back" part of the callback program. The setup file for callback must be properly configured at the host. Callback makes the call to the terminal and then has mgetty run a login on that port. Mgetty by itself (as of early 1998) is only for dial−in calls but there is work being done to incorporate callback features into it and thus make it able to dial−out. As of early 1999 it didn't seem to have been done.

11.4 Terminal Server Connection

One use for terminal servers is to connect many terminals (or modems) to a high speed network which connects to host computers. Of course the terminal server must have the computing power and software to run network protocols so it is in some ways like a computer. The terminal server may interact with the user and ask what computer to connect to, etc. or it may connect without asking. One may sometimes send jobs to a printer thru a terminal server.

A PC today has enough computing power to act like a terminal server for text terminals except that each serial port should have its own hardware interrupt. PC's only have a few spare interrupts for this purpose and since they are hard−wired you can't create more by software. A solution is to use an advanced multiport serial card which has its own system of interrupts (or on lower cost models, shares one of the PC's interrupts between a number of ports). See Serial−HOWTO for more info about such cards. If such a PC runs Linux with getty running on many serial ports it might be thought of as a terminal server. It is in effect a terminal server if it is linked to other PC's over a network and if its job is mainly to pass thru data and handle the serial port interrupts every 14 (or so) bytes. Software called "radius" is sometimes used.

Terminal servers evolved to serve more than just terminals. They also serve PC's which emulate terminals, and were sometimes connected to a bank of modems connected to phone lines. With the advent of 56k digital modems that require a digital connection to service an incoming phone call, a digital interface to the telephone company was needed. This (and more) is provided today by "remote access servers" which have replaced the terminal server. Instead of many individual telephone line cables connected to a terminal server, one now finds just a few cables with many digitized telephone calls on a each cable (multiplexed). The multitude of connectors needed for large numbers of terminals or modems is no longer present on a remote access server and thus the successor to the terminal server can't readily serve text−terminals anymore.

11.5 Connector and Adapter Types

A connector is more−or−less permanently attached to the end of a cable or to a hardware unit. There are two basic types of connectors used in serial communications: 1. DBxx with pins (such as DB25) and 2. modular telephone−style connectors.

An adapter looks about like a connector but it has two ends. It is just like a cable that is so short that there is no cable part left at all −−just different connectors on each end is all that remains. The adapter just plugs in on each side. It allows two incompatible connectors to mate with each other by going in between them. Sometimes the purpose of the adapter is to interchange wires. Obviously, one may use a special cable (perhaps homemade) as a substitute for an adapter.

Sex of connector/adapters

Connectors (or one side of adapters) are either male or female. The connectors that have pins are male and the ones that have sockets (sometimes also called pins) are female. For modular connectors, the ones with exposed contacts are plugs while the ones with internal contacts (not easy to see) are jacks. Plugs are male; jacks are female.

Types of adapters

There are three basic types of adapters: null modem, gender changers and port adapters. Some adapters perform more than one of these three functions.

- null modem adapter: Reroutes wires. Like a null modem cable.
- gender changer: Changes the sex of a cable end. Two connectors of the same sex can now connect (mate) with each other.
- port adapter: Goes from one type of connector to another (DB9 to DB 25, etc.)

DB connectors

(For how to install a DB connector on the ends of a cable see Installing DB Connectors.) These come in 9 or 25 pins. The EIA−232 specs. call for 25 pins but since most of these pins are not used on ordinary serial ports, 9 pins is sufficient. See DB9−DB25 for the pin−out. The pins are usually numbered if you look closely enough or use a magnifying glass.

RJ modular connectors

These look like modern telephone connectors but are sometimes not compatible with telephone connectors. See also Installing RJ Connectors. They may be 6, 8, or 10 conductor.

6−conductors (including MMJ look−alikes)

RJ11/14 is a 4−6 conductor telephone plug. A look−alike is a MMJ connector (6−conductor) used on later model VT (and other) terminals. It's sometimes referred to as a DEC−423. MMJ has an offset tab and is not compatible with RJ11/14 (unless the tab is cut off). However, some connectors have been made that are compatible with both MMJ or RJ11/14. Since MMJ connectors are both hard to find and expensive some people have forced a RJ11/14 (6 conductor) to fit MMJ by filing off a tab with a file. The MMJ pin−out is: 1−DTR, 2−TXD, 3−TXD GND, 4−RXD GND, 5−RXD, 6−DSR.

A standard MMJ null−modem cable has a MMJ connector at each end. It connects to the PC using a MMJ−to−DB adapter. This adapter plugs into a DB (say 25 pin) connector on the back of the PC and the MMJ connecter plugs into it. If you don't have such an adapter, you can make a custom cable with a MMJ (or filed RJ) connector on one end and a DB connector on the other end.

The standard null−modem cable with two MMJ (or RJ11/14) connectors will connect: 1−6, 2−5, and 3−4. Note that such a cable supports DTR/DSR flow control which is not supported (yet) by Linux. Making up your own standard 6−conductor null−modem cable is very simple if you understand that the ordinary 4−conductor telephone cable from the wall to your telephone, used in hundreds of millions of homes, is also a null−modem cable. Find one and wire your cable the same way.

If you lay such a cable (or your terminal null−modem cable) flat on the floor (with no twists) you will note that both plugs on the ends have their gold contacts facing up (or both facing down). Although it's symmetrical, it is also null− modem if you think about it a bit. One may put a few such cables together with inline couplers and everything works OK because each inline coupler is also a null−modem adapter. Two null−modem devices in series result in a straight−thru connection.

Here's a custom cable diagram (by Mark Gleaves) for connecting MMJ to a 9−pin serial port using RTS/CTS flow control:

```
  DEC MMJ                            Linux PC DB9
Pin  Signal                           Signal  Pin
===  ======                           ======  ===
 1    DTR -----------------------|---> DSR     6
                                 |---> CTS     8
 2    TxD ---------------------------> RxD     2
 3    SG (TxD)--------------------|--- SG      5
 4    SG (RxD)--------------------|
 5    RxD <--------------------------- TxD     3
 6    DSR <--------------------------- RTS     7
                                  |--- DTR     4
                                  |--> CD      1
                                       RI      9
```

8−conductors and 10−conductors

RJ45 and RJ48 are usually 8−conductor modular telephone plugs. However some are 10−conductor and are allegedly wider and will not mate with 8−conductor ones. They are used for both flat telephone cable and round twisted pair cable. The cable end of the connector may be different for round and flat cable and both RJ45 and RJ48 may be 8 or 10 conductor so make sure you get the right one. RJ48 has an extra tab so that a RJ48 plug will not push into a RJ45 jack (but a RJ45 plug will mate with a RJ48 jack). They're used on some multiport serial cards and networks. Heres the pin numbers for an 8−conductor:

```
  Plug                         Jack
  (Looking at the end          (Looking at the cavity
   end of a cable)              in a wall)
     .__________.                 .__________.
     | 87654321 |                 | 12345678 |
     |__.    .__|                 |__.    .__|
```

```
             |____|                                  |____|
```

11.6 Making or Modifying a Cable

Buy or make ?

You may try to buy a short, null modem cable. Just a "modem cable" will not work. Null modem cables are often labeled as serial printer cables (but serial printers are not very popular today and neither are the cables). Unfortunately, they will probably not work for hardware flow control (until Linux supports DTR flow control, possibly in 2001). Make sure the connectors on the cable ends will fit the connectors on your computer and terminal.

But if you need longer cables to connect up terminals or need hardware flow control, how do you get the right cables? The right ready−made cables may be difficult to find (you might find them by searching the Internet), especially if you want to use a minimum (say 4) of conductors. One option is to get them custom made, which is likely to be fairly expensive although you might find someone to make them at prices not too much higher than ready−made cable (I did).

A low−cost alternative is to buy used cables (if you can find them). If you get a used terminal, ask if they have a cable for it. Another alternative is to make your own. Even if you get used cables, they may need some changes to the pin wiring. In either case, this may require special tools. Most connectors that come with short cables are permanently molded to the cable and can't be rewired but most custom−made and homemade cables have connectors that can be rewired. One advantage of making your own cable is that the skills you learn will come in handy if a cable breaks (or goes bad) or if you need to make up another cable in a hurry.

Pin numbers

The numbers of the pins should be engraved in the plastic of the connector. Each pin should have a number next to it. You may need a magnifying glass to read them.

Installing DB connectors on cable ends

See DB Connectors for a brief description of them. Unfortunately, most cables one purchases today have molded connectors on each end and can't be modified. Others have connectors which unscrew and can be rewired. If you are making up cable or modifying an existing one then you need to know about pins. There are two types: soldered and crimped.

The crimped pins require a special crimping tool and also need an "insertion/extraction" tool. But once you have these tools, making up and modifying cable may be faster than soldering. If you are connecting two wires to one pin (also needed if you want to jumper one connected pin to another pin) then soldering is faster (for these pins). This is because the crimped pins can only take one wire each while the soldered ones can accept more than one wire per pin.

To insert crimped pins just push them in by hand or with the insertion tool. Using the tool for either insertion of removal first requires putting the tool tip around the wire. The tool tip should completely encircle the wire at the the back of the pin.

Removing a pin with this tool is a little tricky. These directions can be best understood if you have both the tool and wires in front of you as you read this. With the tool tip around the wire insert the tool as far as it will go into the hole (about 1 1/2 cm. Some tools have a mark (such as a tiny hole) on them to indicate how far to insert it. The tool tip should have a tapered gap so that you may get the tip around the wire by starting it in where the gap is wider than the wire. The tool may have 2 tips. The one that is the most difficult to get around the wire is also the one that removes the wire the easiest since it almost completely envelops the wire.

With the tip properly inserted pull on both the tool and the wire with a gentle pull. If it doesn't come out, the tool was likely not inserted correctly so either push it in more or twist it to a different position (or both). Perhaps you should have used the other tip that goes more around the pin. Using this tool, one may readily convert a straight−thru cable to a null−modem cable, etc.

There can be problems using the "insertion/extraction" tool. If the tools will not insert on the back of the pin, it could be that the pin was not neatly crimped to the wire and is sort of square where it should be round, etc. If a pin starts to come out but will not pull out all the way, the pin may be bent. Look at it under a magnifying glass. Straightening a pin with needle−nose pliers may damage the gold plating but you may have to straighten it to remove it. Sometimes a stuck pin may be pushed out with a thick screwdriver blade tip (or the like) but if you push too hard you may gouge the plastic hole or bend the pin:.

Don't try soldering unless you know what you're doing or have read about how to do it.

Installing RJ connectors

These are telephone modular connecters one type of which is used for most ordinary telephones. But there are many different types (see RJ Modular Connectors).

These are not easy to reuse. You might be able to pull the wires out, push in something wedged that would lift up the gold−colored contacts and reuse the connector. There are special crimping tools used to install them; a different tool for each type.

If you don't have a crimping tool, installation is still possible (but difficult) using a small screwdriver (and possibly a hammer). Push in the cable wires and then push each gold−colored contact down hard with a small screwdriver that will just fit between the insulating ridges between the contacts. You may damage it if you fail to use a screwdriver with a head almost the same thickness as the contacts or if the screwdriver slips off the contact as you are pushing it down. You may also use a small hammer to pound on the screwdriver (push first by hand).

Be sure to not hurt the "remove lever" on the connecter when you push in the contacts. Don't just set it down on a table and push in the contacts. Instead, put a shim (about 1 mm thick) that fits snugly in the crevice between the lever and the body. For such a shim you may use thick cardboard, several calling cards, or wood. Since the bottom of the connector (that you will put on the table) isn't level (due to the "remove lever), make sure that the table top has something a little soft on it (like a sheet of cardboard) to help support the non−level connector. Even better would be to put another 1mm shim under the first 6mm of the connector, supporting it just under where you see the contacts. A soft tabletop wouldn't hurt either. Another method (I've never done this) is to hold the connector in a vice but be careful not to break the connector.

As compared to using a crimping tool, installing it per above takes a lot longer and is much more prone to errors and failure but it's sometimes more expedient and a lot cheaper than buying a special tool if you only have one or two connectors to install.

12. Set−Up (Configure) in General

12.1 Intro to Set−Up

Configuring (Set−Up) involves both storing a configuration in the non−volatile memory of the terminal, and putting commands in start−up files (on you hard disk) that will run each time the computer is powered on (or possibly only when the run−level changes). This section gives an overview of configuring and covers the configuring of the essential communication options for both the terminal and the computer. The next two major sections cover in detail the configuration of the terminal (see Terminal Set−Up and the computer (see Computer Set−Up (Configure) Details.

12.2 Terminal Set−Up (Configure) Overview

When a terminal is installed it's necessary to configure the physical terminal by saving (in its non−volatile memory which is not lost when the terminal is powered off) the characteristics it will have when it is powered on. You might be lucky and have a terminal that has already been set−up correctly for your installation so that little or no terminal configuration is required.

There are two basic ways of configuring a terminal. One is to sit at the terminal and go thru a series of set−up menus. Another is to send escape sequences to it from the host computer. Before you can send anything to the terminal (such as the above escape sequences), its Communication Interface) options such as the baud rate must be set up to match those of the computer. This can only be done by sitting at the terminal since the communications must be set up right before the computer and the terminal can "talk" to each other. See Terminal Set−Up.

12.3 Computer Set−Up (Configure) Overview

Besides possibly sending escape sequences from the computer to configure the terminal, there is the configuring of the computer itself to handle the terminal. If your lucky, all you need to do is to put a "getty" command in the /etc/inittab file so that a "login:" prompt will be sent to the terminal when the computer starts up. See Getty (in /etc/inittab) for details. this for the computer.

The computer communicates with the terminal using the device driver software (part of the kernel). The serial device driver has a default configuration and is also partly (sometimes fully) configured by the getty program before running "login" at each terminal. However, additional configuration is sometimes needed using programs named "stty" and "setserial". These programs (if needed) must be run each time the computer starts up since this configuration is lost each time the computer powers down. See Computer Set−Up (Configure) Details.

12.4 Many Options

There are a great many configuration options for you to choose from. The communication options must be set right or the terminal will not work at all. Other options may be set wrong, but will cause no problem since the features they set may not be used. For example, if you don't have a printer connected to the terminal it makes no difference how the printer configuration parameters are set inside the terminal. This last statement is not 100% correct. Suppose that you have no printer but the compute (by mistake) sends the terminal a command to redirect all characters (data) from the computer to the printer only. Then nothing will display on the screen and your terminal will be dead. Some terminals have a configuration option to inform the terminal that no printer is attached. In this case the terminal will ignore any command to redirect output to the "printer" and the above problem will never happen. However, this doesn't help much since there are many other erroneous commands that can be sent to your terminal that will really foul things up. This is likely to happen if you send the terminal a binary file by accident.

In some cases a wrong setting will not cause any problem until you happen to run a rare application program that expects the terminal to be set a certain way. Other options govern only the appearance of the display and the terminal will work fine if they are set wrong but may not be as pleasant to look at.

Some options concern only the terminal and do not need to be set at the computer. For example: Do you want black letters on a light background? This is easier on the eyes than a black background. Should a key repeat when held down? Should the screen wrap when a line runs off the right end of the screen? Should keys click?

12.5 Communication Interface Options

Some of these communication settings (options) are for both the terminal and the computer and they must be set exactly the same for both: speed, parity, bits/character, and flow control. Other communication options are only set at the terminal (and only a couple of these are essential to establish communications). Still others such as the address and interrupt (IRQ) of the physical port ttyS2 are set only at the computer using the "setserial" command. Until all of the above essential options are compatibly set up there can be no satisfactory serial communication (and likely no communication at all) between the terminal and the computer. For the terminal, one must set these options manually by menus at each terminal (or by using some sort of special cartridge at each terminal). The host computer is configured by running commands each time the computer is powered up (or when people log in). Sometimes the getty program (found in the /etc/inittab file) which starts the login process will take care of this for the computer. See Getty (in /etc/inittab)

The settings for both the computer and the terminal are:

- Speed (bits/second)
- Parity
- Bits per Character
- Flow Control

Some essential settings for the terminal alone are:

- Port Select
- Set communication to full duplex (=FDX on Wyse terminals)

If the Getty (in /etc/inittab) program can't set up the computer side the way you want, then you may need to use one (or both) of the Stty & Setserial commands.

Speed

These must be set the same on both the terminal and the computer. The speed is the bits/sec (bps or baud rate). Use the highest speed that works without errors. Enabling flow control may make higher speeds possible. There may be two speeds to set at the terminal: Transmit and Receive, sometimes abbreviated T and R. Usually they are both set the same since stty in Linux doesn't seem to have the option yet of setting them differently. (There is an option to do this with the "stty" command but it seems to actually set them both the same.) Common speeds are 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, ... The slower speeds (like 600) are for printers and hard−copy terminals.

Parity & should you use it ?

For a definition see Parity Explained. Parity−disabled is often the default. To enable parity, you must both enable it and then select either even or odd parity. It probably makes no difference if it's odd or even. For terminals there are sometimes settings for both transmit and receive parity. You should set both of these the same since stty at the computer doesn't permit setting them differently. The PC serial port usually can't support different parities either. Some terminal are unable to set receive parity and will simply always ignore received parity bits. On some older terminals if you use 8−data−bits per byte then parity will not work since there is no room in the hardware for the extra parity bit.

Should you use parity at all? Parity, while not really necessary, is nice to have. If you don't have parity, then you may get an incorrect letter here and there and wind up trying to correct spelling errors that don't really exist. However parity comes at a cost. First, it's more complicated to set up since the default is usually no parity. Secondly, parity will slow down the speed with which bytes travel over the serial cable since there will be one more bit per byte. This may or may not slow down the effective speed.

For example, a hard−copy terminal is usually limited by the mechanics of the printing process. Increasing the bytes/sec when the computer (its UART chip) is transmitting only results in more flow−control "halt" signals to allow the mechanical printing to catch up. Due to more flow−control waits the effective speed is no better without parity than with it. The situation is similar for some terminals: After you implement parity there may be fewer flow−control waits per unit time resulting in more bits/sec (average). However, due to the added parity bits the bytes/sec (average) stays the same.

One option is to install terminals with no parity. Then if parity errors are noticed, it can be implemented later. To spot possible errors with no parity, look for any spelling errors you don't think you made. If you spot such an error, refresh the screen (retransmit from the computer). If the error goes away, then it's likely a parity error. If too many such errors happen (such as more than one every few hundred screens) then corrective action is needed such as: Enable parity and/or reduce speed, and/or use a shorter/better cable. Enabling parity will not reduce the number of errors but it will tell you when an error has happened.

Just the opposite policy is to initially enable parity. Then if no parity errors (error symbols on the CRT) are ever seen (over a reasonable period of time, say a month or two) it may be safely disabled.

Bits/Character

This is the character size (the number of data bits per character excluding any parity bit). To use international character sets you need 8 bits. But it's not of much use unless your terminal has the fonts for them. See Character−Sets If you are only going to use ASCII characters, then select 7−bits since it's faster to transmit 7 bits than 8. Some very old terminals only support 7−bit characters.

Which Flow Control (Handshaking) ?

The choice is between "hardware" (for example dtr/cts) or "software" (Xon/Xoff) flow control. (The Adds terminal menu incorrectly use "Xon/Xoff" to mean any kind of flow control.) While hardware flow control may be faster (if the one or two extra wires for it are available in the cable and if the terminal supports it) in most cases Xon/Xoff should work OK. Some people report that they solved disturbing problems (see below) by converting to hardware flow control but software flow control has worked fine at other installations (and for me personally).

If you use software (Xon/Xoff) flow control and have users who don't know about it, then they may accidentally send an Xoff to the host and lock up their terminal. While it's locked, they may type frantically in a vain attempt to unlock it. Then when Xon is finally sent to restore communication, all that was typed in haste gets executed, perhaps with unexpected results. They can't do this with hardware flow control. See Flow Control for an explanation of flow control.

Port select

Since most terminals have two or more connectors on the back, it is usually possible to assign one of these connecters to connect to the host computer and assign another connector to be the printer port. The connector may have a name next to it (inspect it) and this name (such as Aux, Serial 2, or Modem) may be assigned to either be the main host connection or the printer connection (or the like).

12.6 Quick Attempt

While all the above may seem overly complex, to get a terminal working is often fairly simple. The Quick Install section describes a simple way to try to do this. But if that doesn't work or if you want to make the display look better and perform better, more reading will be needed.

---

13. Terminal Set−Up (Configure) Details

Except for the next subsection on sending escape sequences to the terminal, this section mainly presents the details of setting up the terminal manually by sitting at the terminal and going thru menus. If you haven't already done so, you should read Terminal Set−Up (Configure) Overview. It's best if you have a terminal manual, but even it you don't there is information here on many of the options which you might possibly need to set.

The communication parameters such as its baud rate must always be set up at the terminal since if this is not done there can be no communication with the terminal. Once communication is established you have two choices for doing the rest the terminal configuration. You may continue to configure manually at the terminal and save the results in the terminal's non−volatile memory or you may do this by sending escape sequences to the terminal from the computer each time the terminal is powered on (or the like).

If you know how to set up and save a good configuration inside the terminal it may be the best way. If you don't, you might want to just send the init string from terminfo to your terminal each time you use the terminal. Perhaps doing nothing will still give you a usable terminal. You (or an application program) can always change things by sending certain escape sequences to the terminal.

13.1 Send Escape Sequences to the Terminal

Once the communication interface is established, the rest of the configuration of the terminals may sometimes be done by sending escape sequences to the terminals from the computer. If you have a large number of terminals, it may be worthwhile to write (or locate) a shell script to automatically do this. There may (or may not) be a command you can send to a terminal to tell it to save its current set−up in its non−volatile memory so that it will be present the next time the terminal is powered on.

There is an simple way to send these escape sequences and a complex way. Using the simple way, you never look up escape sequences but issue commands that automatically find an appropriate escape sequence in the terminfo database and send that. Unfortunately, not all the escape sequences which you might want to send are always in the terminfo database. Thus the more complex (but possibly better) way is to directly send escape sequences.

For this complex method you'll need an advanced manual. Old terminal manuals once included a detailed list of escape sequences but newer ones usually don't. To find them you may need to purchase another manual called the "programmers manual" (or the like) which is not supplied with the terminal. A Esc Sequence List for some terminals is on the Internet but it's terse and likely incomplete.

Even without a manual or the like, you may still send commands to configure the terminal by using the programs "tput" and "setterm". See Changing the Terminal Settings. You could just send the terminal an init string from the terminfo entry if the init string sets up the terminal the way want it. See Init String. Unless you plan to have these sequences sent from the computer to the terminal each time the terminal is powered on, you must somehow save the settings in the non−volatile memory of the terminal.

13.2 Older Terminals Set−Up

On older terminals look at the keyboard for labels just above the top row of numeric keys. If they exist, these labels may be what these keys do in set−up mode. Some older terminals may have only one "set−up" menu. Still older ones have physical switches. In some cases not all the switches are well labeled but they may be well concealed. Of course, if you set something with a switch, it's "saved" and there is no need to save the setting in non−volatile memory.

13.3 Getting Into Set−Up (Configuration) Mode

To select options (configure) at the terminal, you must first enter "set−up" mode and then select options (i.e. configure) using menus stored inside the terminal and displayed on the screen. To do this, the terminal does not even need to be connected to a computer. How to get into set−up mode is covered in the terminal's manual, but here's some hints that may help:

If there's a "set−up" key try pressing it. Also try it shifted.

- Wyse: First try the shifted "Select" key; then substitute Ctrl for shifted in all of the above.
- VT, Dorio: F3 may be the set−up key. On VT420 and later models this key may have been programmed to do something else so turn off the power. When you turn on the power again, hit the F3 key as soon as you get an initial screen message.
- IBM: 3151: Ctrl−ScrollLock. 3153: Ctrl−Minus_on_Keypad (or like 3151)

To move around in the set−up menus, try the arrow keys. Use Return, Space, or a special key ("toggle" on old terminals) to select. To exit set−up mode select exit from a menu (or on some older terminals press the set−up key again).

13.4 Communication Options

For the terminal to work at all, speed, parity, :its/character, and communication mode must be set correctly. Incorrect flow control may cause loss and/or corruption of data seen on the screen. The essential communication options were dealt with (for both the terminal and computer) in another section: See Communication Interface. The following list provides some links to that section, as well as some additional communication options set only at the terminal.

* Speed (bits/second) (baud rate): 9600, 19200, etc.
* Parity none, even, odd, mark, space
* Bits per Character {Data}: 7 or 8
* Flow Control: or Handshake {Hndshk}: none, Xon−Xoff, or hardware (DTR, etc).
    * Receiver Handshake {Rcv Hndshk} protects data being Received by the terminal by transmitting flow−control signals to the host.
    * Transmitter Handshake {Xmt Hndshk} is protection of data being Transmitted by the terminal. The terminal receives flow−control signals (and locks/unlocks the keyboard). Includes "Incoming Xon/Xoff".
* number of stop bits: 1 or 2. See Voltage Sequence for a Byte
* Flow control level {Rcv Hndshk Level} {{Xoff at ...}}: Flow control will send "stop" when this number of bytes in the terminal's buffer is exceeded.
* Communication Mode {Comm}: Full Duplex {FDX}, Half Duplex {HDX} {{Local Echo}}, Local Mode {{Online/Local}}
* Transmit Rate (Speed) Limit {Xmt Lim}: limits the transmit rate to the specified cps (chars/sec) even though the baud rate setting may be at a higher speed.
* Function−Key Rate Limit: as above but for function key messages.
* Port Select: Which physical connecter is for the host {Host Port} ?

13.5 Saving the Set−up

Your set−up must be saved in the non−volatile memory of the terminal so that it will be effective the next time you turn on the terminal. If you fail to save it, then the new settings will be lost when you turn off the terminal. Before you go to the trouble of setting up a terminal, make sure that you know how to save the settings. For modern terminals the save command is done via a menu. In some older terminals, only the manual tells how to save. For many of these you press Ctrl−S to save.

13.6 Set−Up Options/Parameters

What follows in this section describes some of the options which are available in the set−up menus of many terminals. Options are also called parameters or features. Many options may be called "modes". Setting options is often called "configuring". Many of these options may also be set by sending certain escape sequences to the terminal. Different models and brands of terminals have various options and the same option may be called by different names (not all of which are given here) Terse names used by Wyse are enclosed in {...}. Names used mostly for VT terminals are enclosed in {{...}}.

13.7 Emulation {Personality} {{Terminal Modes}}

Most modern terminals can emulate several other terminals. The terminal can likely do more if it is set to emulate itself (actually no emulation) {native personality}. Sometimes there are 2 different emulations for the same model of terminal. For example VT220−7 emulates a VT220 with 7−bits/byte while VT220−8 emulates a VT220 with 8−bits/byte (256 possible characters).

Older models of terminals usually have fewer features than newer models. Suppose one wanted to emulate an old terminal but also wanted some of the advanced capabilities of the later model terminal they are sitting at. This is sometimes possible (to some degree). This feature is sometimes called {Enhance} (or Enhanced ??).

13.8 Display Options

Character Cell Size {Char Cell}

This is the size of the cell in which a character fits. It is measured in pixels (=tiny dots). The more dots, the better the resolution. 10x16 is 10 dots wide by 16 dots high (16 rows and 10 columns). Note the notation is inverted as compared to the notation for matrix dimensions which gives rows (height) first.. Also, the character cell includes rows and columns of pixels allocated for the space between adjacent characters so the cell size which defines the boundaries of an actual character may be smaller.

Columns/Lines

Usually 80 columns and 24 or 25 lines. This means that there may be up to 80 characters in a row (line) on the screen. Many terminals have a 132 column option but unless you have a large screen, the tiny characters may be hard to read. {{Set 132 column mode}}. If you set 25 lines, make sure that this is in the terminfo. You should also put "export LINES=25" into /etc/profile and also use: "stty −F /dev/ttySx rows 25". If you don't it might result in a scrolling problem (see Terminal doesn't scroll

Cursor

The cursor may be set to appear as a rectangle (= block) {Blk}. Other options are underline {Line} or blinking. I prefer non−blinking {Steady} block since it's big enough to find quickly but there is no distractive blinking. If you set it invisible (an option on some terminals) it will disappear but new letters will appear on the screen as you type at the invisible cursor.

Display Attributes (Magic Cookies)

Display Attributes may either be magic cookies or be attribute bytes assigned to each character. For magic cookies, there is a limit to their extent: Are they in effect to the end of the line or to the end of the page? It's best to use attribute bytes (which could actually be half−bytes = nibbles).

Display Control Characters {Monitor}

May be called various names such as "Display Controls". When off (normal) it's "Interpret Controls". When set on, you see the escape sequences from the host (which you normally never see on the screen). So that these sequences may be viewed in sequence on a line, they are not acted upon (interpreted) by the terminal. Except that a CR LF sequence creates a new line. See Control Codes.

Double Width/Height

Some terminals can have their characters double width and/or double height. This feature is seldom needed. When changing a line to double width (DW) the right half (RH) is pushed off the screen and there is the question of whether or not to delete (erase) it. "Preserve" means to keep the RH of DW lines. When in double height mode, it may be necessary to send each such line twice (the 2nd time down one row) in order to get a double−height line on the screen.

Reverse Video {Display} (Background Light/Dark)

Normal video is light (white, green, amber) letters (foreground) on a dark (black) background. Reverse video {Display Light} is the opposite: black text on a light background. This is easier on the eyes (unless the room is dark).

Status Line

A status line is a line at the top or bottom of the screen that displays info about the application program you are running. It's often highlighted in some way. With a status line enabled, an application can send the terminal a special escape sequence which means that the text that follows is for the status line. However, many applications don't use this feature but instead only simulate a real status line by direct cursor positioning. The ordinary user looking at it doesn't know the difference.

Upon 80/132 Change: Clear or Preserve?

When switching the number of columns from 80 to 132 (or conversely) should the data displayed in the old format be erased (cleared) or preserved? {80/132 Clr} {{Screen Width Change}}. It should make no difference how you set this option since if an application program uses 132 columns, it should set this option appropriately via a control sequence.

13.9 Page Related Options

For a Wyse terminal to be able to access multiple pages of display memory {Multipage} must be set to on.

Page Size

The terminal memory may be divided up into a number of pages. See Pages and Pages (definition) for explanations of pages. You may partition the page memory into a number of pages of selected length. Linux applications don't seem to use pages at present so it shouldn't make much difference how you set this up.

Coupling (of cursor & display)

The terminal memory may be divided up into a number of pages. See Pages and Pages for explanations of pages. When the cursor is moved to a location in video memory not currently displayed (such as another page, or on the same page but to a location not displayed on the screen) should the display change to let one view the new cursor location? If so, this is called "Coupling". For cursor movement within the same page there is "Vertical Coupling" and "Horizontal Coupling". For movement to another page there is "Page Coupling".

13.10 Reporting and Answerback

The terminal will identify itself and its state, or send out a pre−recorded message in response to certain escape sequences.

Answerback Message (String)

You may write a short message during set−up which may optionally be sent to the host at power−up or be sent to the host in response to a request from the host (perhaps the ENQ (inquire) control character).

Auto Answerback

If set, sends the answerback message to the host at power−on without the host asking for it. Do any "getty" processes look for this ??

Answerback Concealed

If set, will never let anyone see the answerback message (except of course the host computer). If it needs to be changed, deselect "answerback concealed" and the formerly concealed message will be destroyed so you then may enter a new message (but you don't get to see the old one).

Terminal ID {ANSI ID}

The terminal send this reply in answer to a request for identity.

13.11 Keyboard Options

Keyclick

When set, pressing any key makes a click (broadcast by a tiny loudspeaker in the keyboard). These clicks annoy some people and I think it's best to set keyclick off.

Caps Lock {Keylock}

When the Caps−Lock key is down, should only the alphabetic keys generate shifted characters? If set to {Caps} or upper−case−only then hitting a number key with the Caps−Lock on will type the number. To get the symbol above the number one must manually hold down the shift key. This is the normal mode. If set to {Shift} then all keys type the shifted character when Caps−Lock is on (hitting the 5 key should type % without holding down Shift, etc.).

Auto Repeat {Repeat}

If a key is held down then that key is repeatedly "typed". This is handy for repeatedly typing the same character to create a line across the page.

Margin Bell

When the cursor is 8 columns away from the right side of the display, a bell is rung (like on an old typewriter). Almost all editors will automatically create a new line if needed (no need to hit the Return key) so this feature is seldom needed.

Remapping the Keys

The code sent to the host when a key is pressed is normally the ASCII code for that key (depends also on Shift and Control key). On some terminals you may make any key send any code you wish. That is, you may completely remap the keyboard by setting up the terminal that way. This may be useful for some foreign languages and Dvorak keyboard layouts, etc. which permit one to type faster. Even for terminals that don't have the feature, there is software to remap the keyboard (and screen also). It's something like a device driver which uses a pseudo terminal. See Character Mapping: mapchan

Corner Key (for Wyse only)

Wyse terminals have a key near the lower left corner which may be set to do various things. Its may be labelled "Funct", "Compose Character", "Alt", "Hold" or "Scroll Lock". Early models don't have all of the following options: When set to {Hold} No−Scroll it stops the flow of data (using flow control) to the terminal. Hitting the key again restores normal flow. When set to {Compose} it permits one to generate a limited number of pre−defined non−Latin characters. When set to Meta, it makes it a meta shift key which sets the high−order bit on each byte. When set to {Funct} (and pressed) any alphanumeric key pressed gets a header (SOH) and trailer (CR) byte framing the ASCII byte code. When set to {Kpd Compose} (and pressed) then typing a decimal number on the numeric keys (followed by "enter") sends out the same number in hexadecimal ??

Numeric Keypad or Arrow Keys Sends

The numeric keypad (the rectangle of mostly numeric keys to the right of the main part of the keyboard) can be set to send special codes which will do special things in certain application programs. Ditto for the arrow keys. There is thus a "normal" mode where they send what is shown on the keycap (or the normal code sequence for an arrow−key) and an "application" mode where a special escape sequence is sent. In some cases there is a "hex" numeric mode which is almost like normal numeric mode except that 6 non−numeric keys send the letters A−F. Thus one may type for example "B36F" on the numeric keypad.

What does shifted−del and shifted−bs send?

Depending on how they're set up shifted−del sometimes sends the control character CAN and shifted backspace sometimes sends DEL.

PC Scan Codes

Many terminals can emulate a PC keyboard by sending PC scancodes (see Keyboard−and−Console−HOWTO) instead of ASCII codes. This would be used if you were directly connected to a PC running Dos. For Wyse set {Keycode} to {Scan}. Emulating the Dec "PCTerm" should do this and more. However, hardly any Linux programs that run via the serial port can accept scancodes. If this is the latest version of this HOWTO, let me know if any programs do this. I think Foxpro can do it. You need to define smsc and rmsc in the terminfo, and perhaps pctrm.

When using scancodes it's best to use hardware flow control since normal software flow control conflicts with some of the codes (??). If you do use software flow control, you must use the XPC type of flow control. It uses 0x65 and 0x67 for on and off characters. It must be set this way both in the terminal and by stty for the PC.

Alternate Characters

Some keys may have alternative letters on them. When keys is set to "Typewriter" they send what they would normally send on a typewriter. When keys is set to something else the alternative characters are sent.

13.12 Meaning of Received Control Codes

Auto New Line {Newline}

In this case "New Line" means a new line starting at the left margin below the current line. In Linux and C "new line" (NL) may have a different meaning: the line−feed control character LF also called new−line or NL. This is because in Linux text files, the LF character means a "new line starts here" so it's labeled NL. Normally, a LF (NL) sent to a terminal only results in the cursor jumping down one line below where is was and does not move the cursor back to the start of this "new line".

If Auto New Line is set, the above "normal" situation is canceled and a physical new line is created on the display upon receiving a LF from the host. This is exactly what one wants in Linux. Except that (when Auto New Line is set) the Return (or Enter) key sends a CR LF sequence to the host (for Wyse and VT100, but for VT420 ??). Since Linux uses LF as a "new line" marker in files, Linux would like only a LF to be sent (and not a CR LF). Thus the "New Line" option is seldom used. Instead, the required translations are made by the serial port device driver by default. It is as if one gave the command "stty onlcr icrnl". But you don't need to do this since it's the default.

Auto Line Feed {Rcv CR}

This is just another type of "Auto New Line". When a CR (carriage return) character is received, a LF (line feed) action is added resulting in a new line being displayed. Since Linux marks the end of lines with LF, this option is not used.

Recognize Del (Wyse Only ??) or Null

If off, the DEL character received by the terminal is ignored. If on the DEL performs a destructive backspace. Null characters are usually ignored in any case. Both DEL and NULL are sometimes used for padding. See Padding

13.13 Where New Text Goes

Line Wrap

Also called "Auto Wrap(around)". What happens when the right edge of the screen is reached (col. 80, etc) and no return character (or the like) has been sent from the host? If Line Wrap is set, then the rest of the line displays on the line below, etc. Otherwise, the rest of the line is lost and is not seen on the screen. Any good application should handle the situation correctly (provided the terminfo knows how Line Wrap is set). Thus even if Line Wrap is not set, the application should either wrap the screen for long lines or provide another way for you to view the cutoff tail of long lines (by use of the arrow keys, etc). But a raw copy command (and other situations) may not do this so it's often best to set line wrap.

For an 80 col. screen, most terminals only wrap if the 81st character from the host is a graphic (printable) character. This allows for the case where 81st character from the host might be "return" or a "newline" (non−graphic characters) which means that the application is handing the wrapping OK and intervention by the terminal is not needed.

Scrolling

Scrolling {Scrl} is where all the lines on the screen move up or down. Its also called "panning" which includes movement sideways. In ordinary scrolling lines roll off the bottom or top of the screen and disappear, and new lines from the host appear at the opposite edge (top or bottom). There are 3 types of this: smooth, jump, or burst. Burst is not really scrolling since its an instant replacement of an old screenfull by a new one (although some lines on the new screen may be from the old screen). Jump is where new lines jump into view one at a time. Smooth {Smth} is where the text moves at a steady speed upward or downward. If the smooth scroll rate is slow enough, one may read the newly visible lines when they are still scrolling (in motion).

Smooth scrolling on slow terminals was once useful since one could continue reading as the display was scrolling. But with higher baud rates, jump scroll is so fast that little time is lost as the new display appears. Since it takes a little longer to read scrolling text than fixed text, it may actually waste more time if smooth scrolling is selected.

If (auto)scrolling {Autoscrl} is disabled, then new text from the host must go somewhere so it is put at the top of the display. If the old text is not erased, the new text merges (nonsensically) into the old. If the old text is erased, then the new text is out of context. So keep (auto)scrolling enabled.

New Page?

See Pages and Pages for explanations of pages. When the current page is full (the last line is finished) should the page scroll, or should a new page be created (leaving the previous page stored in the terminal's display memory). If {Autopage} is set, then a new page is created. Since you are probably not using pages, you should probably set this to off.

13.14 Function Keys

These are the keys labeled F1, F2, etc. On older terminals they may be labeled PF1, PF2, etc. where the P stands for Programmable. Some keyboards have both. One may program (redefine) these keys to send out a string of user−defined bytes. They may often be easily "programmed" using a certain set−up menu {FKey}. On some terminals, one may also specify where this string is sent to when the key is pressed. In "normal" mode, pressing the key is just like typing the string at the keyboard. In "local" mode pressing the key sends it to the terminal (just like if the terminal was in local mode). This may be used to send escape sequences to the terminal so as to configure it in a special way. In "remote" mode, the string is always sent out the serial port to the host computer (even if the terminal is in local mode).

13.15 Block Mode Options

Some options are only for the case of Block Mode. This option is powerful since it provides forms and takes load off the host computer by transmitting in bursts. But it's more complicated to set up and is thus not used too much.

Forms Display

In block mode some regions of the screen are for the text of forms and are thus write−protected "Prot" {WPRT}. Options may set the characters in these regions to appear dim, reverse video {WPRT Rev}, and/or underlined {WPRT Undrln}. {WPRT Intensity} may be set to dim, normal, or even blank (invisible)

Send Entire Block ?

Should write−protected text (the original text in the form) be sent to the host upon transmission of a block: {Send All} or is write−protected text also read−protected: {Send Erasable}

Region to Send

Should the entire screen be sent or just the scrolling region? {Send Area}. Should the sending stop when the current cursor position is reached? If {Xfer Term} is set to Cursor, only the data on the screen up to the cursor is sent.

Block/Page terminator

What is the termination symbol to be appended to a block of data? {Blk End} or at the end of a page {Send Term}ination.

13.16 Locks

There are various types of Locks. One is the Locked keyboard due to flow control. See Keyboard Lock Another lock {Feature Lock} is that which prohibits the host computer from changing the terminal set−up by sending certain escape sequences to the terminal. Placing such a lock may result in unexpected behavior as application programs send escape sequences to the terminals that are ignored. Not all set−up parameters lock. Unless you have a good reason to do so, you should not enable such locking.

A Function Key lock will prohibit the computer from redefining what a programmable function key sends. You may want to use this if you have something important programmed into the function keys.

13.17 Screen Saver {Scrn Saver}

Also called "CRT Saver". This turns off (or dims) the screen after the terminal is not used for a period of time. It prolongs the life of the screen and may save some energy. Hitting any key will usually restore the screen and may "execute" that key so it's best to hit the shift−key, etc.

13.18 Printer

For Wyse, if there is no {Printer Attached} set it to Off. It's not essential to do this, but if you do it any escape sequence to send text to the printer (instead of the terminal) will be ignored.

Setting up the printer port is about the same (usually simpler) as setting up the communications on the main port. There are a couple of options specific to the printer. Is the printer a serial or parallel printer? If it's parallel it should be designated as such in setup and connected to the parallel port on the terminal (if there is one). Should a FF (form feed) be sent to the printer at the end of a print job? If {Print Term} is set to FF, this will happen.

---

14. Computer Set−Up (Configure) Details

There are various files to edit to set up the computer for terminals. If you're lucky, you'll only need to edit /etc/inittab. One does this by editing at the console (or from any working terminal).

14.1 Getty (in /etc/inittab)

In order to have a login process run on a serial port when the computer starts up (or switches run levels) a getty command must be put into the /etc/inittab file. Running getty from the command line may cause problems (see If getty run from command line: Programs get stopped to see why). Getty GETs a TTY (a terminal) going. Each terminal needs its own getty command. There is also at least one getty command for the console in every /etc/inittab file. Find this and put the getty commands for the real terminals next to it. This file may contain sample getty lines for text terminals that are commented out so that all you need to do is to uncomment them (remove the leading #) and change a few arguments.

The arguments which are permitted depend on which getty you use:
Two gettys best for directly connected terminals are:

1. agetty (sometimes just called getty): Very easy to set up. No config files. See Agetty
2. getty (part of getty_ps)

Two gettys best for modems (avoid for directly connected terminals) are:

1. mgetty: the best one for modems; works for terminals too but inferior
2. uugetty: for modems only; part of the getty_ps package

Simple gettys to use if you don't use a real text−terminal. Most Linux users use one of these at their monitor:

1. mingetty
2. fbgetty

Your Linux distribution should come with either ps_getty or agetty for text−terminals. Unfortunately, they often just call it "getty" so you may need to determine which one you have since the arguments you put after it in /etc/inittab differ. Debian uses agetty (in the util−linux package). RedHat uses ps_getty. As a last resort you might check the source code (usually in /sbin). ps_getty has /etc/gettydefs embedded in the source code. To search for it, go to /sbin and type:
`strings getty | grep getty`
If getty is actually agetty the above will result in nothing. However if you have agetty typing:
`getty −h`
should show the options [−hiLmw].

If you don't have the getty you want check other distributions and the `alien` program to convert between RPM and Debian packages. The source code may be downloaded from Getty Software.

If you are not using modem control lines (for example if you only use the minimum number of 3 conductors: transmit, receive, and common signal ground) you should let getty know this by using a "local" flag. The format of this depends on which getty you use.

If getty run from command line: Programs get stopped

You should normally run getty from inside `/etc/inittab` and not from the command line or else some programs running on the terminal may be unexpectedly suspended (stopped). Here's why (skip to the next section if the why is not important to you). If you start getty for say ttyS1 from the command line of another terminal, say tty1, then it will have tty1 as its "controlling terminal" even though the actual terminal it runs on is ttyS1. Thus it has the wrong controlling terminal. But if it's started inside the inittab file then it will have ttyS1 as the controlling terminal (correct).

Even though the controlling terminal is wrong, the login at ttyS1 works fine (since you gave ttyS1 as an argument to getty). The standard input and output are set to ttyS1 even though the controlling terminal remains tty11. Other programs run at ttyS1 may inherit this standard input/output (which is connected to ttyS1) and everything is OK. But some programs may make the mistake of trying to read from their controlling terminal (tty1) which is wrong. Now tty1 may think that these programs are being run in the background by tty1 so an attempt to read from tty1 (it should have been ttyS1) results in stopping the process that attempted to read. (A background process is not allowed to read from its controlling terminal.). You may see a message something like: "`[1]+ Stopped`" on the screen. At this point you are stuck since you can't interact with a process which is trying to communicate with you via the wrong terminal. Of course to escape from this you can go to another terminal and kill the process, etc.

Agetty (may be named getty)

An example line in /etc/inittab:

```
S1:23:respawn:/sbin/getty −L 19200 ttyS1 vt102
```

S1 is from ttyS1. 23 means that getty is run upon entering run levels 2 or 3. respawn means that if getty is killed, it will automatically start up (respawn) again. /sbin/getty is the getty command. The −L means Local (ignore modem control signals). −h (not shown in the example) enables hardware flow control (same as stty crtscts). 19200 is the baud rate. ttyS1 means /dev/ttyS1 (COM2 in MS−DOS). vt102 is the type of terminal and this getty will set the environment variable TERM to this value. There are no configuration files. Type "init q" on the command line after editing getty and you should see a login prompt.

Agetty's detection of parity

The `agetty` program will auto−detect any parity set inside the terminal. Except it will not work if you use 8−bit data bytes with 1−bit parity. See Agetty's parity with 8−bit data bytes. If you use `stty` to set parity, `agetty` will automatically unset it since it wants the parity bit to come thru as if it was a data bit. This is because it needs to get the last bit (possibly a parity bit) as you type your login−name so that it can auto−detect parity. Thus if you use parity, enable it only at the terminals and let `agetty` auto−detect it and set it at the computer. If your terminal supports received parity, the login prompt will look garbled until you type something so that getty can detect the parity. The garbled prompt will deter visitors, etc. from trying to login. That could be just what you want.

There is sometimes a problem with auto detection of parity. This happens because after you first type your login name, `agetty` uses the `login` program to finish logging you in. If the first login attempt fails, `login` runs again to handle all future attempts at logging in (including the type−in of your login−name). The problem is that only agetty can detect parity while the `login` program doesn't detect parity. So if for some reason you wind up in the `login` program and the correct parity hasn't yet been detected, you're in trouble since the `login` program can't detect the parity. With wrong parity, the `login` program can't correctly read what you type and you can't log in. If your terminal supports received parity, you will continue to see a garbled screen.

One may get into this "login loop" in various ways. Suppose you only type a single letter or two for your login name and then hit return. If these letters are not sufficient for parity detection, then login runs before parity has been detected. Sometimes this problem happens if you don't have the terminal on and connected when agetty first starts up. If you get stuck in this "login" loop a solution is to just wait about a minute until agetty runs again due to "timeout".

Agetty's parity with 8−bit data bytes

Unfortunately, agetty can't detect this parity. It (as of late 1999) has no option for disabling the auto−detection of parity and thus will detect incorrect parity. The result is that the login process will be garbled and parity will be set wrong. Thus it doesn't seem feasible to try to use 8−bit data bytes with parity.

getty (part of getty_ps)

(Most of this is from the old Serial−HOWTO by Greg Hankins)
For this getty one needs to both put entries in a configuration file and add an entry in `/etc/inittab`. Here are some example entries to use for your terminal that you put into the configuration file `/etc/gettydefs`.

```
# 38400 bps Dumb Terminal entry
DT38400# B38400 CS8 CLOCAL # B38400 SANE -ISTRIP CLOCAL #@S @L login: #DT38400

# 19200 bps Dumb Terminal entry
DT19200# B19200 CS8 CLOCAL # B19200 SANE -ISTRIP CLOCAL #@S @L login: #DT19200

# 9600 bps Dumb Terminal entry
DT9600# B9600 CS8 CLOCAL # B9600 SANE -ISTRIP CLOCAL #@S @L login: #DT9600
```

Note that the DT38400, DT19200, etc. are just labels and must be the same that you use in `/etc/inittab`.

If you want, you can make `getty` print interesting things in the login banner. In my examples, I have the system name and the serial line printed. You can add other things:

```
@B    The current (evaluated at the time the @B is seen) bps rate.
@D    The current date, in MM/DD/YY.
@L    The serial line to which getty is attached.
@S    The system name.
@T    The current time, in HH:MM:SS (24-hour).
@U    The number of currently signed-on users.  This is  a
      count of the number of entries in the /etc/utmp file
      that have a non-null ut_name field.
@V    The value of VERSION, as given in the defaults file.
To display a single '@' character, use either '\@' or '@@'.
```

When you are done editing `/etc/gettydefs`, you can verify that the syntax is correct by doing:

```
linux# getty -c /etc/gettydefs
```

Make sure there is no other `getty` or `uugetty` config file for the serial port that your terminal is attached to such as (`/etc/default/{uu}getty.ttyS`*N* or `/etc/conf.{uu}getty.ttyS`*N*), as this will probably interfere with running `getty` on a terminal. Remove such conflicting files if they exits.

Edit your `/etc/inittab` file to run `getty` on the serial port (substituting in the correct information for your environment − port, speed, and default terminal type):

```
S1:23:respawn:/sbin/getty ttyS1 DT9600 vt100
```

Restart `init`:

```
linux# init q
```

At this point, you should see a login prompt on your terminal. You may have to hit return to get the terminal's attention.

mgetty

The "m" stands for modem. This program is primarily for modems and as of mid 1999 doesn't always work very well for text−terminals. It's poorly documented for terminals and you may need to wade thru much documentation for modems in order to figure out how to use it for terminals. Look at the last lines of /etc/mgetty/mgetty.config for an example of configuring it for a terminal. It will only support software (Xon/Xoff) flow control (used by many terminals) if you recompile it. This will be hopefully be fixed in the future. It would be nice to use the same getty for terminals as for modems but mgetty needs a little fixing to fill the bill.

14.2 Stty & Setserial

There is both a "stty" command and a "setserial" command for setting up the serial ports. Some (or all) of the needed stty settings can be done via getty and there may be no need to use setserial so you may not need to use either command. These two commands (stty and setserial) set up different aspects of the serial port. Stty does the most while setserial configures the low−level stuff such as interrupts and port addresses. To "save" the settings, these commands must be written in certain files (shell scripts) which run each time the computer starts up. Distributions of Linux often supply a shell script which runs `setserial` but seldom supply one which runs `stty` since on seldom need it.

14.3 Setserial

This part is in 3 HOWTOs: Modem, Serial, and Text−Terminal. There are some minor differences, depending on which HOWTO it appears in.

Introduction

Don't ever use `setserial` with Laptops (PCMCIA). `setserial` is a program which allows you to tell the device driver software the I/O address of the serial port, which interrupt (IRQ) is set in the port's hardware, what type of UART you have, etc. Since theres a good chance that the serial ports will be automatically detected and set, many people never need to use `setserial`. In any case setserial will not work without either serial support built into the kernel or loaded as a module. The module may get loaded automatically if you (or a script) tries to use setserial.

Setserial can also show how the driver is currently set. In addition, it can be made to probe the hardware and try to determine the UART type and IRQ, but this has severe limitations. See Probing. Note that it can't set the IRQ or the port address in the hardware of PnP serial ports (but the plug−and−play features of the serial driver can do this).

If you only have one or two built−in serial ports, they will usually get set up correctly without using setserial. Otherwise (or if there are problems with the serial port) you will likely need to deal with setserial. Besides the man page for `setserial`, check out info in `/usr/doc/setserial.../` or `/usr/share/doc/setserial`. It should tell you how setserial is handled in your distribution of Linux.

`Setserial` is often run automatically at boot−time by a start−up shell−script for the purpose of assigning IRQs, etc. to the driver. Setserial will only work if the serial module is loaded (or if the equivalent was compiled into your kernel). If the serial module gets unloaded later on, the changes previously made by `setserial` will be forgotten by the kernel. But recent (2000) distributions may contain scripts that save and restore this. If not, then `setserial` must be run again to reestablish them. In addition to running via a start−up script, something akin to `setserial` also runs earlier when the serial module is loaded (or the like). Thus when you watch the start−up messages on the screen it may look like it ran twice, and in fact it has.

Setserial can set the time that the port will keep operating after it's closed (in order to output any characters still in its buffer in main RAM). This is needed at slow baud rates of 1200 or lower. It's also needed at higher speeds if there are a lot of "flow control" waits. See "closing_wait" in the man pg. If your serial port is Plug−and−Play you may need to consult other HOWTOs such as Plug−and−Play or Serial.

Setserial does not set either IRQ's nor I/O addresses in the serial port hardware itself. That is done either by jumpers or by plug−and−play. You must tell setserial the identical values that have been set in the hardware. Do not just invent some values that you think would be nice to use and then tell them to setserial. However, if you know the I/O address but don't know the IRQ you may command setserial to attempt to determine the IRQ.

You can see a list of possible commands by just typing `setserial` with no arguments. This fails to show you the one−letter options such as −v for verbose which you should normally use when troubleshooting. Note that setserial calls an IO address a "port". If you type:

```
setserial −g /dev/ttyS*
```

you'll see some info about how that device driver is configured for your ports. Note that where it says `"UART: unknown"` it probably means that no uart exists. In other words you probably have no such serial port and the other info shown about the port is meaningless and should be ignored. If you really do have such a serial port, setserial doesn't recognize it and that needs to be fixed.

If you add −a to the option −g you will see more info although few people need to deal with (or understand) this additional info since the default settings you see usually work fine. In normal cases the hardware is set up the same way as "setserial" reports, but if you are having problems there is a good chance that "setserial" has it wrong. In fact, you can run "setserial" and assign a purely fictitious I/O port address, any IRQ, and whatever uart type you would like to have. Then the next time you type "setserial ..." it will display these bogus values without complaint. They will also be officially registered with the kernel as seen by the "scanport" command. Of course the serial port driver will not work correctly (if at all) if you attempt to use such a port. Thus when giving parameters to "setserial" anything goes. Well almost. If you assign one port a base address that is already assigned (such as 3e8) it will not accept it. But if you use 3e9 it will accept it. Unfortunately 3e9 is already assigned since it is within the range starting at base address 3e8. Thus the moral of the story is to make sure of your data before assigning resources with setserial.

While assignments made by setserial are lost when the PC is powered off, a configuration file may restore them (or a previous configuration) when the PC is started up again. In newer versions, what you change by setserial gets automatically saved to a configuration file. In older versions, the configuration file only changes if you edit it manually so the configuration remains the same from boot to boot. See Configuration Scripts/Files

Probing

With appropriate options, `setserial` can probe (at a given I/O address) for a serial port but you must guess the I/O address. If you ask it to probe for /dev/ttyS2 for example, it will only probe at the address it thinks ttyS2 is at (2F8). If you tell setserial that ttyS2 is at a different address, then it will probe at that address, etc. See Probing

The purpose of this is to see if there is a uart there, and if so, what its IRQ is. Use "setserial" mainly as a last resort as there are faster ways to attempt it such as wvdialconf to detect modems, looking at very early boot−time messages, or using `pnpdump --dumpregs`. To try to detect the physical hardware use the −v (verbose) and `autoconfig` command to `setserial`. If the resulting message shows a uart type such as 16550A, then you're OK. If instead it shows "`unknown`" for the uart type, then there is supposedly no serial port at all at that I/O address. Some cheap serial ports don't identify themselves correctly so if you see "`unknown`" you still might have a serial port there.

Besides auto−probing for a uart type, setserial can auto−probe for IRQ's but this doesn't always work right either. In one case it first gave the wrong irq but when the command was repeated it found the correct irq. In versions of setserial >= 2.15, the results of your last probe test may be saved and put into the configuration file `/etc/serial.conf` which will be used next time you start Linux. At boot−time when the serial module loads (or the like), a probe for UARTs is made automatically and reported on the screen. But the IRQs shown may be wrong. The second report of the same is the result of a script which usually does no probing and thus provides no reliable information as to how the hardware is actually set. It only shows configuration data someone wrote into the script or data that got saved in /etc/serial.conf.

It may be that two serial ports both have the same IO address set in the hardware. Of course this is not permitted but it sometimes happens anyway. Probing detects one serial port when actually there are two. However if they have different IRQs, then the probe for IRQs may show IRQ = 0. For me it only did this if I first used `setserial` to give the IRQ a ficticious value.

Boot−time Configuration

When the kernel loads the serial module (or if the "module equivalent" is built into the kernel) then only `ttyS{0-3}` are auto−detected and the driver is set to use only IRQs 4 and 3 (regardless of what IRQs are actually set in the hardware). You see this as a boot−time message just like as if `setserial` had been run.

To correct possible errors in IRQs (or for other reasons) there may be a file somewhere that runs `setserial` again. Unfortunately, if this file has some IRQs wrong, the kernel will still have incorrect info about the IRQs. This file should run early at boot−time before any process uses the serial port. In fact, your distribution may have set things up so that the setserial program runs automatically from a start−up script at boot−time. More info about how to handle this situation for your particular distribution might be found in file named "setserial..." or the like located in directory /usr/doc/ or /usr/share/doc/.

Before modifying a configuration file, you can test out a "proposed" `setserial` command by just typing it on the command line. In some cases the results of this use of `setserial` will automatically get saved in /etc/serial.conf when you shutdown. So if it worked OK (and solved your problem) then there's no need to modify any configuration file. See New configuration method using /etc/serial.conf.

Configuration Scripts/Files

Your objective is to modify (or create) a script file in the /etc tree that runs setserial at boot−time. Most distributions provide such a file (but it may not initially reside in the /etc tree). In addition, setserial 2.15 and higher often have an /etc/serial.conf file that is used by the above script so that you don't need to directly edit the script that runs setserial. In addition just using setserial on the command line (2.15+) may ultimately alter this configuration file.

So prior to version 2.15 all you do is edit a script. After 2.15 you may need to either do one of three things: 1. edit a script. 2. edit `/etc/serial.conf` or 3. run "setserial" on the command line which may result in `/etc/serial.conf` automatically being edited. Which one of these you need to do depends on both your particular distribution, and how you have set it up.

Edit a script (required prior to version 2.15)

Prior to setserial 2.15 (1999) there was no /etc/serial.conf file to configure setserial. Thus you need to find the file that runs "setserial" at boot time and edit it. If it doesn't exist, you need to create one (or place the commands in a file that runs early at boot−time). If such a file is currently being used it's likely somewhere in the /etc directory−tree. But Redhat <6.0 has supplied it in /usr/doc/setserial/ but you need to move it to the /etc tree before using it. You might use "locate" to try to find such a file. For example, you could type: locate "*serial*".

The script `/etc/rc.d/rc.serial` was commonly used in the past. The Debian distribution used `/etc/rc.boot/0setserial`. Another file once used was `/etc/rc.d/rc.local` but it's not a good idea to use this since it may not be run early enough. It's been reported that other processes may try to open the serial port before rc.local runs resulting in serial communication failure. Today it's most likely in /etc/init.d/ but it isn't normally intended to be edited.

If such a file is supplied, it should contain a number of commented−out examples. By uncommenting some of these and/or modifying them, you should be able to set things up correctly. Make sure that you are using a valid path for `setserial`, and a valid device name. You could do a test by executing this file manually (just type its name as the super−user) to see if it works right. Testing like this is a lot faster than doing repeated reboots to get it right.

For versions >= 2.15 (provided your distribution implemented the change, Redhat didn't) it may be more tricky to do since the file that runs setserial on startup, /etc/init.d/setserial or the like was not intended to be edited by the user. See New configuration method using /etc/serial.conf.

If you want setserial to automatically determine the uart and the IRQ for ttyS3 you would add something like:

```
/sbin/setserial  /dev/ttyS3 auto_irq skip_test autoconfig
```

Do this for every serial port you want to auto configure. Be sure to give a device name that really does exist on your machine. In some cases this will not work right due to the hardware. If you know what the uart and irq actually are, you may want to assign them explicitly with "setserial". For example:

```
/sbin/setserial /dev/ttyS3 irq 5 uart 16550A  skip_test
```

New configuration method using /etc/serial.conf

Prior to setserial version 2.15, the way to configure setserial was to manually edit the shell−script that ran setserial at boot−time. See Edit a script (after version 2.15: perhaps not). Starting with version 2.15 (1999) of `setserial` this shell−script is not edited but instead gets its data from a configuration file: `/etc/serial.conf`. Furthermore you may not even need to edit serial.conf because using the "setserial" command on the command line may automatically cause serial.conf to be edited appropriately.

This was intended to make it so that you don't need to edit any file in order to set up (or change) setserial so it will do the right thing each time that Linux is booted. But there are serious pitfalls because it's not really "setserial" that edits serial.conf. Confusion is compounded because different distributions handle this differently. In addition, you may modify it so it works differently.

What often happens is this: When you shut down your PC the script that runs "setserial" at boot−time is run again, but this time it only does what the part for the "stop" case says to do: It uses "setserial" to find out what the current state of "setserial" is and puts that info into the `serial.conf` file. Thus when you run "setserial" to change the serial.conf file, it doesn't get changed immediately but only when and if you shut down normally.

Now you can perhaps guess what problems might occur. Suppose you don't shut down normally (someone turns the power off, etc.) and the changes don't get saved. Suppose you experiment with "setserial" and forget to run it a final time to restore the original state (or make a mistake in restoring the original state). Then your "experimental" settings are saved.

If you manually edit serial.conf, then your editing is destroyed when you shut down because it gets changed back to the state of setserial at shutdown. There is a way to disable the changing of serial.conf at shutdown and that is to remove "###AUTOSAVE###" or the like from first line of serial.conf. In at least one distribution, the removal of "###AUTOSAVE###" from the first line is automatically done after the first time you shutdown just after installation. The serial.conf file will hopefully contain some comments to help you out.

The file most commonly used to run setserial at boot−time (in conformance with the configuration file) is now /etc/init.d/setserial (Debian) or /etc/init.d/serial (Redhat), or etc., but it should not normally be edited. For 2.15 Redhat 6.0 just had a file /usr/doc/setserial−2.15/rc.serial which you have to move to /etc/init.d/ if you want setserial to run at boot−time.

To disable a port, use `setserial` to set it to "uart none". The format of /etc/serial.conf appears to be just like that of the parameters placed after "setserial" on the command line with one line for each port. If you don't use autosave, you may edit /etc/serial.conf manually.

BUG: As of July 1999 there is a bug/problem since with ###AUTOSAVE### only the setserial parameters displayed by "setserial −Gg /dev/ttyS*" get saved but the other parameters don't get saved. Use the −a flag to "setserial" to see all parameters. This will only affect a small minority of users since the defaults for the parameters not saved are usually OK for most situations. It's been reported as a bug and may be fixed by now.

In order to force the current settings set by setserial to be saved to the configuration file (serial.conf) without shutting down, do what normally happens when you shutdown: Run the shell−script `/etc/init.d/{set}serial stop`. The "stop" command will save the current configuration but the serial ports still keep working OK.

In some cases you may wind up with both the old and new configuration methods installed but hopefully only one of them runs at boot−time. Debian labeled obsolete files with "...pre−2.15".

IRQs

By default, both ttyS0 and ttyS2 will share IRQ 4, while ttyS1 and ttyS3 share IRQ 3. But actually sharing serial interrupts (using them in running programs) is not permitted unless you: 1. have kernel 2.2 or better, and 2. you've complied in support for this, and 3. your serial hardware supports it. See Serial−HOWTO: Interrupt sharing and Kernels 2.2+.

If you only have two serial ports, ttyS0 and ttyS1, you're still OK since IRQ sharing conflicts don't exist for non−existent devices.

If you add an internal modem and retain ttyS0 and ttyS1, then you should attempt to find an unused IRQ and set it both on your serial port (or modem card) and then use setserial to assign it to your device driver. If IRQ 5 is not being used for a sound card, this may be one you can use for a modem. To set the IRQ in hardware you may need to use isapnp, a PnP BIOS, or patch Linux to make it PnP. To help you determine which spare IRQ's you might have, type "man setserial" and search for say: "IRQ 11".

14.4 Stty

Introduction

`stty` does much of the configuration of the serial port but since application programs (and the getty program) often handle it, you may not need to use it much. It's handy if your having problems or want to see how the port is set up. Try typing ``stty −a'' at your terminal/console to see how it's now set. Also try typing it without the −a (all) for a short listing which shows how it's set different than normal. Don't try to learn all the setting unless you want to become a serial guru. Most of the defaults should work OK and some of the settings are needed only for certain obsolete dumb terminals made in the 1970's.

`stty` is documented in the man pages with a more detailed account in the info pages. Type `"man stty"` or `"info stty"`.

Whereas `setserial` only deals with actual serial ports, stty is used both for serial ports and for virtual terminals such as the standard Linux text interface at a PC monitor. For the PC monitor, many of the stty settings are meaningless. Changing the baud rate, etc. doesn't appear to actually do anything.

Here are some of the items stty configures: speed (bits/sec), parity, bits/byte, # of stop bits, strip 8th bit?, modem control signals, flow control, break signal, end−of−line markers, change case, padding, beep if buffer overrun?, echo what you type to the screen, allow background tasks to write to terminal?, define special (control) characters (such as what key to press for interrupt). See the `stty` man or info page for more details. Also see the man page: `termios` which covers the same options set by stty but (as of mid 1999) covers features which the stty man page fails to mention.

With some implementations of getty (getty_ps package), the commands that one would normally give to stty are typed into a getty configuration file: /etc/gettydefs. Even without this configuration file, the getty command line may be sufficient to set things up so that you don't need stty."')

One may write C programs which change the stty configuration, etc. Looking at some of the documentation for this may help one better understand the use of the stty command (and its many possible arguments).

Serial−Programming−HOWTO is useful. The manual page: termios contains a description of the C−language structure (of type termios) which stores the stty configuration in computer memory. Many of the flag names in this C−structure are almost the same (and do the same thing) as the arguments to the stty command.

Using stty at a "foreign" terminal

Using `stty` to configure the terminal that you are currently using is easy. Doing it for a different (foreign) terminal or serial port may be tricky. For example, let's say you are at the PC monitor (tty1) and want to use `stty` to deal with the serial port ttyS2. Prior to about 2000 you needed to use the redirection operator "<". After 2000 (provided your version of setserial is >= 1.17 and stty >= 2.0) there is a better method using the −F option. This will work when the old redirection method fails. Even with the latest versions be warned that if there is a terminal on ttyS2 and a shell is running on that terminal, then what you see will likely be deceptive and trying to set it will not work. See Two interfaces at a terminal to understand it.

The new method is ``stty −F /dev/ttyS2 ..." (or −−file instead of F). If ... is −a it displays all the stty settings. The old redirection method (which still works in later versions) is to type ``stty ... </dev/ttyS2". If the new method works but the old one hangs, it implies that the port is hung due to lack of a modem control line from being asserted. Thus the old method is still useful for troubleshooting. See the following subsection.

Old redirection method

Here's a problem with the old redirection operator (which doesn't happen if you use the newer −F option instead). Sometimes when trying to use stty, the command hangs and nothing happens (you don't get a prompt for a next command even after hitting <return>). This is likely due to the port being stuck because it's waiting for one of the modem control lines to be asserted. For example, unless you've set "clocal" to ignore modem control lines, then if no CD signal is asserted the port will not open and stty will not work for it (unless you use the newer −F option). A similar situation seems to exist for hardware flow control. If the cable for the port doesn't even have a conductor for the pin that needs to be asserted then there is no easy way to stop the hang.

One way to try to get out of the above hang is to use the newer −F option and set "clocal" and/or "crtscts". If you don't have the −F option then you may try to run program on the port that will force it to operate even if the control lines say not to. Then hopefully this program might set the port so it doesn't need the control signal in the future in order to open: clocal or −crtscts. To use "minicom" to do this you have to reconfigure minicom for another ttyS, etc, and then exit it and restart it. Since you then have to reconfigure minicom again, it may be simpler to just reboot the PC.

The old redirection method makes ttyS2 the standard input to stty. This gives the stty program a link to the "file" ttyS2 so that it may "read" it. But instead of reading the bytes sent to ttyS2 as one might expect, it uses the link to find the configuration settings of the port so that it may read or change them. Some people tried to use ``stty ... > /dev/ttyS2" to set the terminal. This will not do it. Instead, it takes the message normal displayed by the stty command for the terminal you are on (say tty1) and sends this message to ttyS2. But it doesn't change any settings for ttyS2.

Two interfaces at a terminal

When using a shell (such as bash) with command−line−editing enabled there are two different terminal interfaces (what you see when you type stty −a). When you type in modern shells at the command line you have a temporary "raw" interface (or raw mode) where each character is read by the command−line−editor as you type it. Once you hit the <return> key, the command−line−editor is exited and the terminal interface is changed to the nominal "cooked" interface (cooked mode) for the terminal. This cooked mode lasts until the next prompt is sent to the terminal. Note that one never gets to type anything to this cooked mode but what was typed in raw mode becomes cooked mode as soon as one hits the <return> key.

When a prompt is sent to the terminal the terminal goes from "cooked" to "raw" mode (just like it does when you start an editor since you are starting the command−line editor). The settings for the "raw" mode are based only on the basic settings taken from the "cooked" mode. Raw mode keeps these setting but changes several other settings in order to change the mode to "raw". It is not at all based on the settings used in the previous "raw" mode. Thus if one uses stty to change settings for the raw mode, such settings will be lost as soon as one hits the <return> key at the terminal that has supposedly been "set".

Now when one types stty to look at the terminal interface, one may either get a view of the cooked mode or the raw mode. You need to figure out which one you're looking at. It you use stty from another terminal to deal with a terminal that is displaying a command line, then you will see the raw mode settings. Any changes made will only be made to the raw mode and will be lost when someone presses <return> at the terminal you tried to "set". But if you type a stty command at your terminal (without using < for redirection) and then hit <return> it's a different story. The <return> puts the terminal in cooked mode. Your changes are saved and will still be there when the terminal goes back into raw mode (unless of course it's a setting not allowed in raw mode).

This situation can create problems. For example, suppose you corrupt your terminal interface and to restore it you go to another terminal and "stty −F dev/ttyS1 sane" (or the like) to restore it. It will not work! Of course you can try to type "stty sane ..." at the terminal that is corrupted but you can't see what you typed. All the above not only applies to dumb terminals but to virtual terminals used on a PC Monitor as well as to the terminal windows in X. In other words, it applies to almost everyone who uses Linux.

Luckily, when you start up Linux, any file that runs stty at boot−time will likely deal with a terminal (or serial port with no terminal) that has no shell running on it so there's no problem for this special case.

Where to put the stty command ?

Should you need to have `stty` set up the serial interface each time the computer starts up then you need to put the `stty` command in a file that will be executed each time the computer is started up (Linux boots). It should be run before the serial port is used (including running getty on the port). There are many possible places to put it. If it gets put in more than one place and you only know about (or remember) one of those places, then a conflict is likely. So make sure to document what you do.

One place to put it would be in the same file that runs setserial when the system is booted. The location is distribution and version dependent. It would seem best to put it after the setserial command so that the low level stuff is done first. If you have directories in the /etc tree where every file in them is executed at boot−time (System V Init) then you could create a file named "stty" for this purpose.

14.5 Terminfo & Termcap (brief)

See Terminfo and Termcap (detailed) for a more detailed discussion of termcap. Many application programs that you run use the terminfo (formerly termcap) data base. This has an entry (or file) for each model or type (such as vt100) of terminal and tells what the terminal can do, what codes to send for various actions, and what codes to send to the terminal to initialize it.

Since many terminals (and PC's also) can emulate other terminals and have various "modes" of operation, there may be several terminfo entries from which to choose for a given physical terminal. They usually will have similar names. The last parameter of getty (for both agetty and getty_ps) should be the terminfo name of the terminal (or terminal emulation) that you are using (such as vt100).

The terminfo does more than just specify what the terminal is capable of doing and disclose what codes to send to the terminal to get it to do those things. It also specifies what "bold" will look like (will it be reverse video or will it be high intensity, etc.), what the cursor will look like, if the letters will be black, white, or some other color, etc. In PC terminology these are called "preferences". It also specifies initialization codes to send to the terminal (analogous to the init strings sent to modems). Such strings are not automatically sent to the terminal by Linux. See Init String. If you don't like the way the display on the screen looks and behaves you may need to edit (and then update) the terminfo (or termcap) file. See Terminfo Compiler (tic) for how to update.

14.6 Setting TERM and TERMINFO

These are two environment variables for terminals: TERM and TERMINFO, but you may not need to do anything about them. TERM must always be set to the type of the terminal you are using (such as vt100). If you don't know the type (name) see What is the terminfo name of my terminal ?. TERMINFO contains the path to the terminfo data base, but may not be needed if the database is in a default location (or TERMINFO could be set automatically by a file that comes with your distribution of Linux). You may want to look at Compiled database locations.

Fortunately, the getty program usually sets TERM for you just before login. It just uses the terminal type that was specified on getty's command line (in /etc/inittab). This permits application programs to find the name of your terminal and then look up the terminal capabilities in the terminfo data base. See TERM Variable for more details on TERM.

If your terminfo data base can't be found you may see an error message about it on your terminal. If this happens it's time to check out where terminfo resides and set TERMINFO if needed. You may find out where the terminfo database is by searching for a common terminfo file such as "vt100" using the "locate" command. Make sure that your terminal is in this database. An example of setting TERMINFO is: export TERMINFO=/usr/share/terminfo (put this in /etc/profile or the like). If the data for your terminal in this data base is not to your liking, you may need to edit it. See Terminfo & Termcap (brief).

What is the terminfo name of my terminal ?

You need the exact name in order to set the TERM environment variable or to give to `getty`. The same name should be used by both the termcap and terminfo databases so you only need to find it once. A terminal usually has alias names but if more than one name is shown, use the first one.

To find it, try looking at the /etc/termcap... file (if you have it). If not, then either look at the terminfo trees (see Compiled database locations) or try to find the terminfo source code file (see Source−code database locations.

14.7 Rarely Needed /etc/ttytype File

The configuration file /etc/ttytype is used to map /dev/ttySn's to terminal names per terminfo. tset uses it, but if the TERM environment variable is already set correctly, then this file is not needed. Since the Linux getty sets TERM for each tty, you don't need this file. In other Unix−like systems such as FreeBSD, the file

/etc/ttys maps ttys to much more, such as the appropriate getty command, and the category of connection (such as "dialup"). An example line of Linux ttytype: vt220 ttyS1

14.8 Login Restrictions

By default, the root user may not login from a terminal. To permit this you must create (or edit) the file /etc/securetty per the manual page "securetty". To restrict logins of certain users and/or certain terminals, etc. edit /etc/login.access (this replaces the old /etc/usertty file ??). /etc/login.def determines if /etc/securetty is to be used and could be edited so as to make /etc/securetty not needed (or not used). /etc/porttime restricts the times at which certain ttys and users may use the computer. If there are too many failed login attempt by a user, that user may be prohibited from ever logging in again. See the man page "faillog" for how to control this.

14.9 Run Command Only If TERM=my_term_type

Sometimes there are commands that one wants to execute at start−up only for a certain type of terminal. To do this for the stty command is no problem since one uses the redirection operator < to specify which terminal the command is for. But what about shell aliases or functions? You may want to make a function for the ls command so it will color−code the listing of directories only on color terminals or consoles. For monochrome terminals you want the same function name (but a different function body) which will use symbols as a substitute for color−coding. Where to put such function definitions that are to be different for different terminals?

You may put them inside an "if" statement in /etc/profile which runs at startup each time one logs on. The conditional "if" statement defines certain functions, etc. only if the terminal is of a specified type.

Example for ls Function

While much of what this if statement does could be done in the configuration file for dircolors, here's an example for the case of the bash shell:

```
if [ "$TERM" = linux ]; then
    eval `dircolors`;
elif [ "$TERM" = vt220 ]; then
    ls () { command ls −F $* ; }# to export the function ls():
    declare −xf ls
else echo "From /etc/profile: Unknown terminal type $TERM"
fi
```

14.10 Character Mapping: mapchan

There is a program named "mapchan" which will map characters (bytes) typed at a terminal keyboard (input) into different characters per a user−supplied mapping table. It can also map characters which are sent to the screen (output). This is nice for remapping the keyboard for foreign language alphabets. Most distributions don't seem to supply it (let me know if any do). Source code by Yura Kalinichenko (Ukraine) is at http://www.kron.vinnica.ua/free/download/

15. Terminfo and Termcap (detailed)

15.1 Intro to Terminfo

Terminfo (formerly Termcap) is a database of terminal capabilities and more. For every (well almost) model of terminal it tells application programs what the terminal is capable of doing. It tells what escape sequences (or control characters) to send to the terminal in order to do things such as move the cursor to a new location, erase part of the screen, scroll the screen, change modes, change appearance (colors, brightness, blinking, underlining, reverse video etc.). After about 1980, many terminals supported over a hundred different commands (some of which take numeric parameters).

One way in which terminfo gives the its information to an application program is via the "ncurses" functions that a programmer may put into a C program. For example, if a program wants to move the cursor to row 3, col 6 it simply calls: move(3,6). The move() function (part of ncurses) knows how to do this for your terminal (it has read terminfo). So it sends the appropriate escape sequence to the terminal to make this particular move for a certain terminal. Some programs get info directly from a terminfo files without using ncurses. Thus a Linux package that doesn't require ncurses may still need a terminfo file for your terminal.

The terminfo abbreviations are usually longer than those of termcap and thus it's easier to guess what they mean. The manual pages for terminfo are more detailed (and include the old termcap abbreviations). Also, the termcap entries had a size limitation which is not present for terminfo. Thus, unless you are already committed to working with termcap, it's suggested you use terminfo.

15.2 Terminfo Database

Introduction

The terminfo database is compiled and thus has a source part and a compiled part. The old termcap database has only a source part but this source can, by a single command, be both converted to terminfo source and then compiled. Thus you may get by without having any terminfo source since the termcap source can create the compiled terminfo database. To see a display of the database for the terminal you're now using (including a PC monitor) type "infocmp" and you should see the source terminfo "file" for it.

To see if your terminal (say vt100) is in the terminfo data base type "locate vt100". If you need to find the terminfo name for your terminal, explore the listing of files in the compiled database or see What is the terminfo name of my terminal ?

Where is the database located ?

Compiled database locations

Typing "locate vt100" may show /usr/lib/terminfo/v/vt100, /usr/share/terminfo/v/vt100, /home/.../.terminfo/v/vt100, and/or /etc/terminfo/v/vt100. All these are possible locations of the compiled terminfo files. Although the /etc/terminfo directory is not a standard location for it, having a few terminal types there could be useful in case the /usr directory is not accessible. For example /usr could be on a separate disk or partition that failed to mount. Normally, programs that use your main terminfo data base are able to find it if it's in at least one of the locations mentioned above. Otherwise the environment variable TERMINFO may be set to the path to this database. Example: TERMINFO=/usr/share/terminfo

For the Debian Distribution of Linux, several commonly used terminals (including the monitor−console) are in the ncurses−term package. These are put into /etc/terminfo/. All of the terminals in the database are in the ncurses−bin package and go into /usr/share/terminfo/.

If the compiled terminfo is in more than one location, everything is usually OK until someone installs new terminfo files (from a newer distribution, from the net, by editing the old one, etc.). Each new terminfo file needs replace all the existing older copies of that file (at various locations) unless you abolish redundant locations. If you don't ensure this gets done, then some application programs could wind up still finding and using the old (and possibly buggy) terminfo data that sill exists in a "possible" location. Setting the environment variable TERMINFO to the up−to−date location (as mentioned above) would help avoid this problem.

Source−code database locations

While the source−code file may not be installed on your computer there's another way to get the source−code if you have the compiled code. Just use the "`infocmp`" command.

The source code file (for all terminals0 may be /etc/termcap and/or terminfo.src (or another name). See the man pages: terminfo(5) or termcap(5) for the format required to create (or modify) these source files. The file terminfo.src may be in various locations on your computer or it may not be included with your Linux distribution. Use the `locate` command to try to find it. It is available on the web at
http://www.tuxedo.org/terminfo or
http://download.sourceforge.net/mirrors/OpenBSD/src/share/termtypes/termtypes.master

Terminfo Compiler (tic)

The data in the source files is compiled with the "tic" program which is capable of converting between termcap format and terminfo format. Thus you can create a compiled terminfo data base from termcap source. The installation program which was used to install Linux probably installed the compiled files on your hard disk so you don't need to compile anything unless you modify /etc/termcap (or terminfo.src). "tic" will automatically install the resulting compiled files into a terminfo directory ready to be used by application programs. Which location it's installed in depends on ... See "man tic" for the explanation.

Look at Your Terminfo

It's a good idea to take a look at the terminfo entry for the terminal you are using (source code of course) and read the comments. A quick way to inspect it without comments is to just type "infocmp". But the comments may tell you something special about the terminal such as how you need to set it up so that it will work correctly with the terminfo database.

Deleting Data Not Needed

In order to save disk space, one may delete all of the database except for the terminals types that you have (or might need in the future). Don't delete any of the termcaps for a "Linux terminal" (the console) or the xterm ones if you use X−Windows. The terminal type "dumb" may be needed when an application program can't figure out what type of terminal you are using. It would save disk space if install programs only installed the terminfo for the terminals that you have and if you could get a termcap for a newly installed terminal over the Internet in a few seconds.

15.3 Bugs in Existing Terminfo Files (and Hardware)

Unfortunately, there are a number of bugs in the terminfo and termcap files. In addition, many of these definitions are incomplete and do not define certain features available on the terminals. Sometimes you can get by without modifying the terminfo but in other cases you need to modify it or possibly use another emulation that has a good terminfo.

The sad state of the supplied terminfo files is due to a number of reasons. One is that during the 1980's when many of them were written (often in termcap format), application programs did not utilize more advanced terminal features. Thus if such feature were not in the termcap (or terminfo) file, no one complained. Today, programs such as vim use "context highlighting" and minicom uses the terminal's graphics character set. These often need more definitions to be added to the old termcap. This may (or may not) have already been done.

Most terminals had hardware bugs (in their firmware) and sometimes these were "fixed" by modifying the termcap. Then the manufacturer might send out replacement chips which would fix the bug. Not all owners would bother to get the replacement chips. Thus there may be 2 or more terminfos for your terminal, depending on what firmware chips it has in it. This situation was often not noted in the termcap and only one termcap may be supplied with Linux. Some hardware bugs which existed for features that were almost never used in the past likely never did get fixed. Also, some reported hardware bugs may never have been fixed since they were not of much significance at the time or because the company went out of business, etc.

15.4 Modifying Terminfo Files

To do this you need a manual for your terminal showing what escape sequences it uses. Newer manuals from the 1990's often don't show this. You also need a terminfo manual (or the like). For example, in order to add graphic capability you must assign values to the terminfo variables: enacs, rmacs, and smacs by editing a source file. Then by using "tic" you may compile it. "tic" should automatically put the compiled terminfo file in the correct directory reserved for it.

If you would like to find a better terminfo entry for a certain terminal than the one supplied, you might try searching the Internet (but what you find could be worse). If your new terminfo entry is better than the old one and it seems stable (you've used it for a while with no problems) then you should send a copy to the maintainer of terminfo as noted at the start of the source file for terminfo (or termcap).

15.5 Init String

Included in the terminfo are often a couple of initialization strings which may be sent to the terminal to initialize it. This may change the appearance of the screen, change what mode the terminal is in, and/or make the terminal emulate another terminal. An initialization string is not automatically sent to the terminal to initialize it. One might expect that the getty program should do this but if it did, one could make a change to the set−up at the terminal and this change wouldn't be happen because the init string would automatically cancel it. Application programs don't seem to initialize either. To do this you must use a command given on the command line (or in a shell script such as /etc/profile) to send the init strings. Such commands are: "tset", "tput init", or "setterm −initialize". Sometimes there is no need to send the init strings since the terminal may set itself up correctly when it is powered on (using options/preferences one has set up and saved in the non−volatile memory of the terminal).

15.6 TERM Variable

The Environment variable TERM should be set to the name of terminal which you are using. If TERM hasn't been set yet and you don't know the name of your terminal see What is the terminfo name of my terminal ?. It is normally set by the terminal_type parameter passed to the getty program (look at it in the /etc/inittab file). This name must be in the Terminfo data base. Just type "set" at the command line to see what TERM is set to (or type: tset −q). At a console (monitor) TERM is set to "linux" which is the PC monitor emulating a fictitious terminal model named "linux". Since "linux" is close to a vt100 terminal and many text terminals are also, the "linux" designation will sometimes work as a temporary expedient with a text terminal.

If more than one type of terminal may be connected to the same port (/dev/tty...) (for example, if there is a switch to permit different terminal types to use the same serial port, or if the port is connected to a modem to which people call in from different types of terminals) then TERM needs to be set each time someone connects to the serial port. There is often a query escape sequence so that the computer may ask the terminal what type it is. Another way is to ask the user to type in (or select) the type of terminal s/he is using. You may need to use tset for this or write a short shell script to handle this.

One way to do this is to use "tset" (see the manual page). tset tries to determine the terminal name of the terminal you are using. Then it looks up the data in terminfo and sends your terminal an init string. It can also set the value of TERM. For example, a user dials in and logs in. The .profile login script is executed which contains within it the following statement: eval `tset −s ?vt100`. This results in: The user is asked if s/he is using a vt100. The user either responds yes or types in the actual terminal type s/he is using. Then tset sends the init string and sets TERM to this terminal name (type).

15.7 Terminfo/Termcap Documents

- manual pages for terminfo(5) (best) and/or termcap(5). The Termcap Manual (2nd ed.) by Richard M. Stallman is a GNU manual which is somewhat obsolete since it doesn't include terminfo.
- the files: terminfo.src and /etc/termcap have info about various versions of termcap files, naming conventions for terminals, and special capabilities code named u6−u9. If you don't have one, go to http://www.tuxedo.org/terminfo
- "Termcap and Terminfo" is a book published by O'Reilly in 1988.

---

16. Using the Terminal

16.1 Intro to Using Terminal

This section is about controlling the terminal−computer interface and/or changing the terminal set−up while using the terminal. It explains (or points to explanations of) how the user of a terminal can control and inspect the interface and how to use various commands provided by the device driver. It does not explain how to use the many application programs, shells or most Linux utilities. Two commands commonly used at the terminal are:

- clear (to clear the screen)
- reset (to reset the terminal
- setterm −reset (alternative for "reset" in case of bug)

16.2 Starting Up the Terminal

Of course the power must be on for the terminal to work. If you don't see a login prompt hit the "return" (or "enter") key a few times. Then type your account name (followed by a return/enter) and your password when prompted for it (also followed by return/enter). Make sure not to type all capital letters. If you do, the computer may think that you have an old terminal that can't send lowercase letters and the serial driver may set itself up to send only capital letters to the terminal.

If nothing happens, make sure that both the host computer and the terminal are OK. If the host computer is shut down (no power) what you type at the terminal keyboard may appear on the screen since the transmit and receive pins at the computer may be connected together resulting in echoing of characters by an "off" computer. If you can't log in when the host computer is running, see Trouble−Shooting.

16.3 Terminal (Serial) Device Driver

When typing at the command line, the shell (such as the Bash shell) is reading what you type and reacting to it. What you type first passes thru the terminal driver part of your operating system. This driver may translate certain characters (such as changing the "return" character generated by the "return" key into a "new−line" character for Linux files). It also recognizes certain control codes which you may type at the keyboard such as ^C to interrupt the execution of a program. It also normally echoes what you type back to the display. Stty may be used to configure how this terminal driver behaves, including disabling some (or all) of its functionality.

16.4 Problems with Editors

There may be some problems with using both emacs and vi on some terminals. A few terminals have no escape key (ESC) so you may need to type Ctrl−[to get ESC.

emacs

If software flow control exists, then the ^S command in emacs will freeze the display. The ^Q command will unfreeze the display. One fix is to map this to another key−press by configuring emacs that way. Some terminals have meta keys although you may need to setup the terminal to create a meta key.

vi and Cursor−Keys

Vi uses the esc−key as a command to exit insert mode. Unfortunately for most terminals the arrow−keys send an an escape sequence (starting with the ESC character) to the host. Vi must distinguish between these two meanings of ESC. A smart vi (such as vim if configured for it) is able to detect the difference by noting the time between the ESC and the next key. If it's a short time, then it's likely that a cursor key was pressed. Use "help cursor−keys" in vim to find out more.

Here's another way to fix this. On VT terminals the left−arrow−key may be either set to send ESC [D or ESC O D. The other arrow keys are similar but use A, B, and C instead of D. If you're having problems, choose ESC [D since the "O" in the other alternative could be interpreted by vi as a command to "Open a line". The "[" should be interpreted by vi to mean that an arrow−key has been pressed. ESC [D will be sent provided "Cursor Key Application Mode" has not been set. ESC [D is normally the default so everything is seemingly OK. Except that many termcaps contain a string (not the init string) which sets what you want to avoid: "Application Mode". Editors may send this string to the terminal when the editor starts up. Now you are in trouble.

This string has the termcap code "ks" (smkx in terminfo) meaning enable the function (and related) keys (including the arrow keys). An application enables these keys by sending the "ks" string to the terminal. Whoever wrote the termcap reasoned that if an application wants to enable these keys, then they should be put into "Application Key Mode" since this is an "application", but you don't want this.

The Linux console has no "ks" string so you can't fall into this trap at the console. For other terminals you may need to edit the termcap (or terminfo) or use another termcap entry. You need to change not only the "ks" string but also the termcap definitions of what they send: kd, kl, kr, ku. Then run tic to install it.

For vim (vi iMproved) there is a way to set it up to work OK with ESC O D (so you don't need to edit termcap): See vim help for "vt100−cursor−keys". You may run "gitkeys" and then press your cursor keys to see what they send but they may be set to send something different when you're in an editor.

16.5 Bugs in Bash

There have been problems with the readline interface to the Bash shell. Bash 2.01 (1998) had a readline interface which was quite corrupted if one used a dumb terminal. This was fixed in later versions. One problem still outstanding is that if certain terminal keys send bytes with the high order bit set to 1, then Bash seems to ignore the terminfo entries for them. I've reported this as a bug in Bash. Other programs such as the vim editor and the lynx browser work OK with such keys.

To work around this problem one may define what these keys should do in Bash by putting such definitions into `/etc/inputrc.` For example, A Wyse 60 will send D0−D3 when the arrow−keys are hit provided the terminal is in "application key mode". After modifying the terminfo to reflect this, they still wouldn't work on the command line in the Bash shell. So I explicitly defined the arrow−keys in `/etc/inputrc` like this:

```
# Arrow keys in 8 bit keypad mode: Sends d0-d4.  -ap means application.
$if term=wy60-25-ap
set enable-keypad on
"\xd0":         backward-char
"\xd1":         forward-char
"\xd2":         next-history
"\xd3":         previous-history
$endif
```

If the terminal is already in "application key mode" there's no need to "set enable−keypad on". enable−keypad will send the terminal the escape sequence named smkx in terminfo (which for wyse60 is \E 3 and makes the arrow keys send D1−D3). Many other application send this without needing to be told to do so.

16.6 Color ls Corruption

If `ls` is corrupting your terminal emulation with the color feature, turn it off. `ls --color`, and `ls --colour` all use the color feature. Some installations have `ls` set to use color by default. Check `/etc/profile`, etc. for `ls` aliases. See Example for ls Function for how to have `ls` do color for the console and do monochrome for terminals.

16.7 Display Freezes (hung terminal)

The symptom of a hung terminal is where what you type doesn't display on the terminal (or in some cases displays but doesn't do anything). If what you type is invisible (or does nothing) type ^Q to restart flow (if flow control stopped it). Hanging may also be due to:
Sent terminal binary or Abnormally exited a program
If you didn't do any of these two, then the program you're running could by buggy or you interaction with it fatally illegal.

If you want to quit the program you were running and you can't do it by the usual methods (some programs have special keys you must hit to exit) then try killing it from another terminal using "top" or "kill". If the process refuses to die, then kill it with signal 9 from top (or use "kill" on the command line). The "9" type of forced exit may leave some temporary files lying around as well as a corrupted interface. If this doesn't work, killing the login shell should result in a startup of getty with a new login prompt. If all else fails, you may need to reboot the computer or even power it down. Just rebooting may not alter the possibly corrupted content of the serial port hardware registers.

People new to Linux may unintentionally press Ctrl−S (^S) (or the "No Scroll" key) which mysteriously freezes the screen (although that is what this key is supposed to do if you use software flow−control). To restore normal screen interaction, press Ctrl−Q (^Q). Note that everything typed during the "freeze" gets executed but you don't see any report of this until you hit ^Q. Thus when it's frozen, don't type anything drastic that might destroy files, etc. One argument for using hardware flow−control is to prevent such freezes.

16.8 Corrupted Terminal Interface

This includes the case of a "frozen display" = "hung terminal" of the previous section.

Symptoms and some fixes

When the display doesn't look right, or when what you type doesn't display correctly (if at all), or nothing happens when you type a command, you may have a corrupted terminal interface. In rare cases when the serial port hardware gets itself corrupted, the only fix may be to cycle power (turn off the PC and reboot). Sometimes logging in again will solve the problem. To do this kill the shell process running on the terminal (or kill getty if it's running). You do this from another terminal. Once killed, a new getty process respawns which hopefully will end the corruption. Recycling power (or resetting) for the terminal may help too.

The corruption may be due to things such as bug in the program you're using, a hardware failure (including an obscure hardware defect that you can normally live with), or possibly an incorrect configuration. If everything was working normally but it suddenly goes bad, it may be that the interface got corrupted by something you did. Three mistakes you might have made to corrupt the interface are:

- Sent terminal binary
- Abnormally exited a program
- Typed ctrl−S by mistake

Sent terminal binary characters

Your terminal will change its characteristics if sent certain escape sequences or control characters. It you inadvertently try to display a binary file, it might by chance contain such sequences which may put your terminal into some strange mode of operation or even make it unusable. Always view or edit a binary file with programs designed for that purpose so that this doesn't happen. Most editors and pagers will handle binary OK so as not to corrupt the interface. Some may display a message telling you that they can't edit binary. But using "cat ...." or "cp .... /dev/tty.." where .... is a binary file, will send the binary to the terminal and likely corrupt things.

Corruption it can also happen when using a communications program where a remote computer may send binary to your screen. There are numerous other ways it can happen so be prepared for it. Even a supposed ASCII file may contain unwanted control codes.

To fix this problem reset the terminal. Type either just "reset" (may be broken) or "setterm −reset" (followed by a <return> of course). You may not be able to see what you're typing. This will send the reset string from the terminfo entry to the terminal. As an alternative to this, if the correct set−up has been saved inside the terminal then pressing a special key(s) (perhaps in setup mode) may restore the settings. Then you might still need to use "tset" to send the init string if you use it to set up your terminal.

Note that the "reset" command appears to be broken for terminals that need "clocal" set since "reset" seems to unset "clocal". In this case instead of fixing the problem, "reset" only compounds it by disabling communication between the terminal and computer. You will likely need to log in again and hope the getty sets "clocal". If you see a login prompt without asking for it, you're in luck. Otherwise see Symptoms and some fixes for how to get a login prompt. You should try out "reset" before you need it and use "setterm −reset" if "reset" logs you out or otherwise doesn't work right. I submitted a bug report in Mar. 2000 so "reset" could be fixed by now.

For the case where you see strange "graphic" characters instead of the normal ones, pressing ^O may switch back to the normal letters. The "reset" command also does this but it resets everything else too.

Symptoms and some fixes

Abnormally exited a program

Large application programs (such as editors) often use the stty command (or the like) in their code to temporarily change the stty configuration when you are running the program. This may put the device driver into "raw" mode so that every character you type goes directly thru to the application program. Echoing by the driver is disabled so that everything you see on the screen comes directly from the application program. Thus many control commands (such as ^C) may not work within such applications.

When you tell such an application to quit, the application program first restores the stty settings to what they were before the application program started. If you abnormally exit the program (you may guess this has happened when what you type no longer displays on the screen) then you may still be in "raw mode" on the command line.

To get out of raw mode and restore the normal stty settings type "stty sane". However, you must type this just after a "return" and end it with a "return". But hitting the "return" key doesn't do the job since the "return" code no longer gets translated to the new−line characters that the shell is waiting for. So just type new−line (^J) instead of "return". The "sane" terminal interface may not be exactly the same as the normal one but it usually works. "stty sane" may also be useful to get out of a corrupted interface due to other causes.

16.9 Special (Control) Characters

A number of control characters which you may type at the keyboard are "caught" by the terminal driver and perform various tasks. To see these control commands type: stty −a and look at lines 2−4. They are tersely explained in the stty manual pages. They may be changed to different control characters or disabled using the stty command. Thus your control characters might be different than those described below. They are used for command−line editing, interrupting, scrolling, and to pass the next character thru transparently.

Command−Line Editing

While the terminal driver has a few commands for command−line editing, some shells have a built−in real editor (such as "readline" in the Bash shell). Such an editor is normally on by default so you don't need to do anything to enable it. If it's available you don't need to learn many of the following commands although they often still work along with the command−line editor. The most important to learn are ^C (interrupt), ^D, and how to stop scrolling.

- Delete−key (shown by stty as ^?) erases the last character
- ^U kills (deletes) the line
- ^W deletes a word backwards
- ^R reprints the line. Useful mainly on hard copy terminals ??

Interrupting (& Quit, Suspend, EOF, Flush)

- ^C interrupts. Exits the program and returns you to the command−line prompt.
- ^/ quits. Same as interrupt ^C but weaker. Also dumps a "core" file (which you likely have no use for) into your working directory.
- ^Z suspends. Stops the program and puts it in the background. Type fg to restart it.
- ^D end of file. If typed at the command−line prompt, exits the shell and goes to where you were before the shell started.
- ^O flush. Not implemented in Linux. Sends output to /dev/null.

Stop/Start Scrolling

If what you want to see scrolls off the bottom of the screen, you may prevent this by sending a "stop" signal (^S or Xoff) to the host (provided Xon−Xoff Flow Control is enabled). Send a "start signal to resume (^Q or Xon). Some terminals have a "No Scroll" key which will alternately send Xoff and Xon or possibly send the hardware flow control signals ?? Here's what ctrl−S (^S) and ctrl−Q (^Q) do:

- ^S stops scrolling (Xoff)
- ^Q resume scrolling (Xon)

If you want to both stop scrolling and quit, use ^C. If you want to stop scrolling to do something else but want to keep the program that was generating the output in memory so you can resume scrolling later, use ^Z suspend.

An alternative scrolling method is to pipe the output thru a pager such as more, less, or most. However, the output might not be standard output but could be error output which the pager doesn't recognize. To fix this you may need to use redirection "2>&1" to get the pager to work OK. It is often simpler to just use ^S and ^Q unless you need to scroll backwards.

At a PC console (emulating a terminal) you may scroll backwards by using Shift−PageUp. This is frequently needed since the scrolling is often too fast to stop using ^S. Once you've scrolled backwards Shift−PageDown will scroll forward again.

Take next character literally

^V sends the next character typed (usually a control character) directly thru the device driver with no action or interpretation. Echoed back are two ASCII characters such as ^C.

16.10 Viewing Latin−1 Files on a 7−bit Terminal

Some "text" files are 8−bit Latin1 (see Character−Sets). If you have a terminal that will not display Latin1 (or don't have the Latin1 character set selected), then a bullet symbol will display as a 7, etc. When viewing manual pages (they are Latin1) you may give the option −7 to man so as to translate the 7's, etc. to something close to a bullet (in ASCII). Are there some pagers that make these translations ??

16.11 Inspecting the Interface

These utility programs will provide information about the terminal interface:

- gitkeys: shows what byte(s) each key sends to the host.
- tty: shows what tty port you are connected to.
- set: shows the value of TERM (the terminfo entry name)
- stty −a: shows all stty settings.
- setserial −g /dev/tty?? (you fill in ??) shows UART type, port address and IRQ number.
- infocmp: shows the current terminfo entry (less comments)

16.12 Changing the Terminal Settings

The terminal settings are normally set once when the terminal is installed using the setup procedures in the terminal manual. However, some settings may be changed when the terminal is in use. You normally would not give any "stty" of "setserial" commands when the terminal is in use as they are likely to corrupt the terminal interface. However, there are changes you may make to the appearance of the terminal screen or to its behavior without destroying the integrity of the interface. Sometimes these changes are made automatically by application programs so you may not need to deal with them.

One direct method of making such changes is to use the setup key (or the like) at the terminal and then use menus (or the like) to make the changes. To do this you may need to be familiar with the terminal. The other 3 methods send an escape sequence from the computer to the terminal to make the changes. These 3 examples show different methods of doing this to set reverse video:

1. setterm −reverse
2. tput −rev
3. echo ^[[7m

setterm

This is the easiest command to use. It uses long options (but doesn't use the normal −− before them). It consults the terminfo database to determine what code to send. You may change the color, brightness, linewrap, keyboard repeat, cursor appearance, etc.

tput

The "tput" command is similar to "setterm" but instead of using ordinary words as arguments, you must use the abbreviations used by terminfo. Many of the abbreviations are quite terse and hard to remember.

echo

In the example "echo ^[[7m" to set reverse video, the ^[is the escape character. To type it type ^V^[(or ^V followed by the escape key). To use this "echo" method you must find out what code to use from a terminal manual or from terminfo or termcap. It's simpler to use setterm or tput if you are typing on the command line. Since "echo ..." will execute faster (since it doesn't do any lookups) it's good for using in shell scripts which run at start−up, etc.

Saving changes

When you turn off the terminal the changes you made will be lost (unless you saved them in non−volatile terminal memory by going into set−up mode and saving it). If you want to use them again without having to retype them, put the commands in a shell script or make it a shell function. Then run it when you want to make the changes. One way to make the changes semi−permanent is to put the commands in a file that runs each time you login or start up the computer.

16.13 Multiple Sessions

The "`screen`" package runs multiple sessions something like virtual terminals on the console: See The Console: /dev/tty?. However, this is not like "pages" (Pages) since the image of the pages are stored in the host computer and not inside the terminal as they are with "pages".

16.14 Logging Out

To log out type either "logout" or "exit". Under some circumstances your request will be refused, but you should be told why. One reason for refusal is if you are not in the same shell that you logged into. Another way to log out is to press ^D. Since ^D is also used for other purposes, you may not want it to log you out. If you set IGNOREEOF in the Bash shell then ^D will no longer log you out.

16.15 Chatting between Terminals, Spying

If two persons logged into terminals on the same host computer want to chat with each other they may use the "write" or the "talk" (better) program. One may prevent anyone (except the superuser) from writing (sending messages) to their terminal by using the "mesg" command.

For spying on what someone else is doing at their terminal use the "ttysnoop" program. In "ttysnoop" the two terminals have the same status and anything typed on either keyboard appears on both screens (in the same location). So if you're really spying and don't want to be detected, you shouldn't type anything.

"ttysnoop" can be used for training with instructor and trainee sitting side−by−side, each at their own terminal. The instructor may watch what the trainee is typing and can correct any mistakes by typing it correctly. The trainee can watch what the instructor types and then try typing it. It's just like they used one terminal with both people having their hands on the keyboard at the same time.

There's one shortcoming to "ttysnoop" and that is that the terminals involved should all be (or emulate) the same type of terminal (such as vt100). Since the "Linux" emulation done by a console (monitor) and the "minicom" emulation are close to vt100 one may use ttysnoop using two PCs, one running "minicom" which emulates a terminal.

There's a non−free program called "DoubleVision" that is something like ttysnoop but does much more. Terminals may be of different types and it can remember and playback sessions on terminals so you can observe what happened in the past. The webpage is at http://www.tridia.com.

16.16 Sharing the Serial Port

If you have another serial device (a printer, modem, etc.) that you want to connect to the same serial port that a terminal is on, then there is more involved than just plugging in the other device. This is mainly because when the getty program or shell is listening on the port, they (or the serial driver) are likely to give a response to any bytes sent to the port. Getty will respawn even if it's killed and will keep sending login prompts to the serial port (which may now be connected to another device). One way to work around this problem is to switch runlevels so that no getty program or shell is running on the port. But if your other use for the port only uses the port for output to an external device, then you may live dangerously by using the port for the other use with the shell or getty (intended for a terminal) simultaneously running on it.

For the runlevel fix, you may create another runlevel in which no getty program runs on the port. Then you enter that new runlevel and use the serial port for something else. To set this up you need to edit `/etc/inittab` and check and/or set the runlevels at which getty runs. For example the line: "`S1:23:respawn:/sbin/getty` ..." means that getty will run (on ttyS1) in runlevels 2 and 3. Now you could have it only run in level 2 (by deleting "3") and then go to runlevel 3 when you want to use the other serial device. Thus to use the serial port for something else, the super−user would type "init 3" to switch to runlevel 3. Type "init 2" to get back to runlevel 2. Note that each runlevel may have a different set of initialization scripts but you can change this if you want, so that the same scripts are run in both runlevels. How the scripts and runlevels work are different for each Linux distribution. In Debian, the explanation of this is in `/usr/doc/sysvinit` but also look in `/usr/share/doc.`

There's also the problem of the stty configuration of the port. When the port is being used for the terminal it has certain configurations but when say "init 3" is used to switch runlevels and disable getty on the port, the original (boot−time) configuration of the port is not restored. You are likely to wind up with the port configured in a "raw" mode. This means that the serial port likely needs to be fully reconfigured by stty, either by a script you write or by the next application which runs on the port. Some applications such as printing from the serial port are not capable of fully reconfiguring (the /etc/printcap can only partially reconfigure). Thus you may need to write a script to do it. The stty configuration of a terminal will be different depending on whether a shell is running on it, whether it's at the "login" prompt, etc. Thus the stty configuration after switching runlevels may vary.

17. Special Uses for a Terminal

17.1 Make a Serial Terminal the Console

This will turn a text−terminal (or emulator PC) into a "serial console". Many messages from the system are normally only sent to the console (the PC monitor). Some of the messages sent to the console at boot−time may also be seen on any terminal after the boot succeeds by typing the command: dmesg. If the boot failed this will not be of any use since the terminal can't work without an operating system. It's possible to modify the Linux kernel so as to make a terminal serve as the console and receive all the messages from Linux intended for the console. Unfortunately, the messages from the BIOS (which display on the monitor when a PC is first started) will not display on this terminal (but still display on the monitor).

Some people do this when they run a PC without a monitor or keyboard. Normally, a PC will not start up without a keyboard and video card but some BIOSs permit it. If you are lucky enough to have such a BIOS that supports "console re−direct" you will likely then need to setup the BIOS using the CMOS menus when you start your PC.

The method of creating a "serial console" depends on your kernel version. In any case serial support needs to be compiled into the kernel and not supplied as a module.

For Kernels 2.2 or higher

The instructions for creating a serial−console are included with source code documentation in the file: serial−console.txt. Normally, the device /dev/console is linked to tty0 (the PC console). For a serial−console you create a new /dev/console by

```
mknod -m 622 console c 5 1
```

You must also put statements regarding the serial−console into /etc/lilo.conf and then run lilo. This is because the equivalent of "setserial" needs to be run early before the kernel is loaded so that the serial−console can display the booting. See the above mentioned documentation and the man page for lilo.conf for more details.

Here is an example `/etc/lilo.conf` file contents (for ttyS0):

```
prompt
timeout=50
boot = /dev/sda
vga = normal # force sane state
install = /boot/boot.b
message = /boot/message
image = /vmlinuz
root = /dev/sda1
label = console
serial = 0,9600n8
append = "console=ttyS0"
```

For Kernels before 2.2

The Linux Journal in April 1997 had an article on patching the Linux kernel. You add a couple of #defines at the start of src/linux/drivers/char/console.c:

```
#define CONFIG_SERIAL_ECHO
#define SERIAL_ECHO_PORT 0x2f8  /* Serial port address  */

The following was not in the Linux Journal article.
```

```
In kernel 2.+ (and earlier ??) you need to also set the baud
rate (unless 9600 is OK).  Find these 2 lines:

serial_echo_outb(0x00, UART_DLM); /* 9600 baud */
serial_echo_outb(0x0c, UART_DLL);

Change 0x0c in the line above (depending on the baud rate you want):

115200 baud: 0x01       19200 baud: 0x06        2400 baud: 0x30
 57600 baud: 0x02        9600 baud: 0x0c        1200 baud: 0x60
 38400 baud: 0x03        4800 baud: 0x18
```

If you currently use the console to select which operating system to boot (using LILO), but would like to do this from a terminal, then you need to add a line to the /etc/lilo.conf file. See the manual page for lilo.conf and search for "serial=".

17.2 Run Linux without a Monitor

One way to do this is to just use a terminal (serial console) for the monitor See Make a Serial Terminal the Console. You will likely still need a video card since most BIOSs require one to get the PC started. Your BIOS may also require a keyboard to get started or it may have an option where you can set the BIOS not to require a keyboard.

If you have a keyboardless terminal, it can also be used as a monitor by use of the ttysnoop program. As of yet, it doesn't work like a console since it doesn't get all the initial boot−time messages. See Use a Keyboardless Terminal as the Monitor

17.3 Use a Keyboardless Terminal as the Monitor

How it works

While you might think that a terminal without a keyboard is useless it is possible to use it as the monitor and type on the PC's own keyboard. This may be done by using the spy program `ttysnoop`. This program permits a person at one terminal to spy on (or snoop) what someone else is typing at another terminal (or the PC console−monitor).

Now suppose you are typing away at the monitor tty1. Imagine that someone with a terminal on ttyS2 is spying on you (with ttysnoop) and has a screen that looks just like your screen. Everything you type at tty1 also displays on ttyS2. Now move the spy terminal (on ttyS2) so it is side−by−side with your monitor (on tty1). Everything you type on the PC keyboard (not the terminal keyboard) will now display on the two screens in front of you: the monitor and the spy terminal. Now just look only at the spy terminal as you type. Disconnect both the monitor and the terminal keyboard since you're not using either. Thus you are now using a keyboardless terminal as a monitor. What a simple but clever solution!

One possible problem (which turns out to be no problem at all) is that normally, the type of spy terminal should be the same as the type of terminal being spied upon. Since the monitor is normally declared as a terminal of type "linux" (which is close to vt100), you might think that the real terminal should also be (or at least emulate) a vt100. Not necessarily so! For example, if you have a wyse60 terminal you simply falsely declare that you have a wyse terminal on tty1 (tell the getty program for tty1 that it's a wyse60).

Here's why you can get away with this. Go back the the scenario where you have both the monitor and the terminal in front of you, both displaying the same thing (or trying to). If the terminal is a wyse60 ttysnoop will be sending it the same bytes as are going to the monitor (via its driver). Unfortunately, some of escape sequence for the monitor (which an application program thinks is of type "linux") will not be recognized by the wyse60 and the wyse60 display will be corrupted. But this can be fixed as follows.

At the monitor, falsely declare that it is a wyse60. Now when you use an application program it sends wyse60 escape sequences to both the monitor and the wyse60 terminal. The situation is now the other way around. The display on the wyse60 is fine but the display on the monitor is corrupted since it's getting wyse60 escape sequences. Since you will not use the monitor (and are about to disconnect it) the corruption is never a problem (you don't see it).

A problem remaining is that ttysnoop should start eary in the boot process so that you get the login prompt on the terminal. If you work out how to do this (and this is the latest version of the HOWTO), let me know.

One way to set it up

This is not the ideal setup since ttysnoop runs so late that the login prompt doesn't appear. This example is for a wyse60 terminal on ttyS2 and the missing monitor/PC−keyboard on tty1. It uses the agetty program for getty as supplied by the Debian distribution. Your getty may require a different format.

In `/etc/inittab:`

```
1:2345:respawn:/sbin/getty 38400 tty1 wyse60 −l /usr/sbin/ttysnoops
```

Note that `ttysnoop/ttysnoops` is a client−server combo so the "s" is not a typo.

In `/etc/snooptab:`

```
# tty       snoopdev      type    execpgm
tty1        /dev/ttyS3    init    /usr/local/bin/sterm
```

In the above, a script named sterm is used to run the stty program. You may not need this or may want to use it for another purpose. Here's the example sterm script in `/usr/local/bin/sterm:`

```
#!/bin/sh
stty rows 24
/bin/login $@
```

---

18. Trouble−Shooting

If you suspect that the problem is a hardware problem, see the Repair and Diagnose section. If it's the keyboard see Keyboards. If the problem involves the serial port itself see the Serial−HOWTO.

Here is a list of possible problems:

- Is the Terminal OK ? Suspect the terminal is defective.
- Display Freezes (hung terminal)
- Types Garbage Characters you type are "foreign" letters/symbols.
- Missing Text Either skips over some text or displays some text OK and hangs.
- All Keys Work Erratically; Must Hit a Key a Few Times
- If getty run from command line: Programs get stopped with message "Stopped"

* Getty Respawning Too Rapidly (console error message)
* Fails Just After Login
* Can't Login but login prompt is OK.
* Garbled Login Prompt
* No Login Prompt

There are two cases where the terminal goes bad. One is when it's been recently working OK and suddenly goes bad. This is discussed in the next sub−section. The other case is where things don't work right just after you install a terminal. For this case you may skip over the next section.

18.1 Terminal Was Working OK

When a formerly working terminal suddenly goes bad it is often easy to find the problem. That's because (except for hardware failures) the problem is likely due to something that you did (or something the software you used did).

The problem may be obvious such as an error message when the terminal is first turned on. If it makes a strange noise it likely needs repair. See Repair & Diagnose. First, think about what has been done or changed recently as it's likely the cause of the problem. Did the problem happen just after new system software was installed or after a change in the configuration?

If the terminal isn't responding correctly (if at all) to what you type to it, you may have a Corrupted Terminal Interface.

18.2 Terminal Newly Installed

If you've just connected up a terminal to a computer per instructions and it doesn't work this section is for you. If a terminal that formerly worked OK doesn't work now then see Terminal Was Working OK If you suspect that the serial port on your computer may be defective you might try running a diagnostic test program on it. At present (June 1998) it seems that Linux doesn't yet have such a diagnostic program so you may need to run diagnostics under MS DOS/Windows. There are some programs to monitor the various serial lines such at DTR, CTS, etc. and this may help. See Serial Monitoring/Diagnostics

One approach is to first see if the the terminal will work by trying to copy a file to the terminal (cp my_file /dev/ttyS?) under the most simple situation. This means with the modem control lines disabled and at a show speed that doesn't need flow control (make sure that any hardware flow control is disabled). If this copy works, then make the situation a little more complicated and see if it still works, etc., etc. When the trouble appears just after you made a change, then that change is likely the source of the trouble. Actually, its more efficient (but more complex) to jump from the simple situation to about half way to the final configuration so that the test eliminates about half of the remaining possible causes of the problem. Then repeat this methodology for the next test. This way it would only take about 10 tests to find the cause out of a thousand possible causes. You should deviate a little from this method based on hunches and clues.

18.3 Is the Terminal OK ?

Many terminals will start up with some words on the screen. If these words convey no error message, it's probably OK. If there is no sign of power (a dark screen, etc.) re−plug in the computer power cord at both ends. Make sure there is power at the wall jack (or at the extension cord end). Try another power cord if you have one. Make sure the terminal is turned on and that its fuse is not blown. A blank (or dim) screen may sometimes be fixed by just turning up the brightness and contrast using knobs or a keyboard key in set−up mode.

A terminal that's been stored for a long time may take a while to "warm up" as the electrolytic capacitors heal themselves under voltage. If it still won't work See Repair & Diagnose for tips on repairing it.

If the terminal starts up OK, but you suspect that something may be wrong with it, go into "local mode" where is works like a typewriter and try typing on it. See Local Mode. Before you have trouble, you should find out if your terminal displays error messages if the hardware is bad. One way to test a terminal for this is to turn it on with the keyboard unplugged and see if it displays an error message.

18.4 Missing Text

If several lines or text displays on the terminal OK and then it stops without finishing (in the middle of a word, etc.) or if big chunks of text are missing, you likely have a problem with flow control. If you can't figure out right away what's causing it, decrease the speed. If that fixes it, it's likely a flow control problem. It may be that flow control is not working at all due to failure to configure it correctly, or due to incorrect cable wiring (for hardware flow control). See Flow Control

If you can type OK at the terminal but when text is sent to the terminal only about 1 in every 16 characters sent gets thru, then you may have given the wrong UART to setserial. This will happen if the port is an obsolete 16550 (or lower) but you've told setserial it's a 16550A or higher.

If single characters are missing, perhaps the serial port is being overrun by too fast a speed. Try a slower baud rate.

If your device is under 1200 baud (probably a very slow old hard−copy terminal or printer) and the text gets truncated, then the problem may be in the serial device driver. See Printing−HOWTO under "Serial devices" on how to fix this.

18.5 All Keys Work Erratically; Must Hit a Key a Few Times

This is where you need to hit a key a few times before it works (and see the letter you typed on the screen). If you type a word, some (or even all) of the letters may be missing on the screen. If letters are missing from a command it doesn't work and even if all letters are present you may need to hit the return−key several times to get the command to execute.

This may be due to two different processes opening the serial port. Both try to read what you type. Sometimes one process (the right one −−perhaps the shell) reads what you type and at other times the other process reads what you type. An example is where the other process is for a serial mouse (such as gpm) which doesn't echo what you type. So another process running on the same ttySx is eating some of what you type. To fix this, use "ps −alx" to see what else is running on your ttySx and kill that process.

You might think that lockfiles would prevent this. But neither the terminal nor the gpm mouse program uses lockfiles. Since others may need to write to your terminal, it's reasonable not to use lockfiles. See Lock−Files in the Serial−HOWTO.

18.6 Getty Respawning Too Rapidly

No modem control signal

If getty can't open and/or use a port because of the lack of a positive modem control voltage on one of the pins, then getty might be killed. Then, per the instructions in inittab, getty respawns and tries again, only to be killed again, etc., etc. You may see an error message like: Getty respawning too fast: disabled for 5 minutes. Instead of "Getty" it may give the label (such as S2) of the line in /etc/inittab from where from where getty was called. Try using the "local" option with getty and/or check the modem control settings and voltages.

No serial support

Linux provides serial support either by selecting it when compiling the kernel or by loading the serial module. This "respawning" error happens if serial support has neither been built into the kernel nor provided by a serial module. You many use the "lsmod" command to see if the serial module is loaded.

Key shorted

Another possible cause of getty respawning too rapidly is if a keyboard key is shorted, giving the same result as if the key was continuously held down. With auto−repeat enabled, this "types" thousands of characters to the login prompt. Look for a screen filled with all the same character (in some cases with 2 or more different characters).

18.7 Fails Just After Login

If you can login OK but then you can't use the terminal it may be because the starting of the login shell has reconfigured the terminal (to an incorrect setting) by a command which someone put into one of the files that are run when you login and a shell starts. These files include /etc/profile and /.bashrc. Look for a command starting with "stty" or "setserial" and make sure that it's correct. Even if it's done OK in one initialization file, it may be reset incorrectly in another initialization file that you are not aware of. Ways to get into the systems to fix it are to use another terminal or console, use a rescue diskette, or type: "linux single" at the lilo prompt which puts you into single user mode without running startup files.

18.8 Can't Login

If you get a login prompt but get no response (or perhaps a garbled response) to your login attempts a possible cause is that the communication is bad one−way from the terminal to the computer. It could be a bad or mis−wired cable/connector. If you're not already using the "local" option with getty, do so to disable the modem control lines. See Getty (in /etc/inittab). You might also disable hardware flow control (stty −crtscts) if it was enabled. If it now works OK then your modem control lines are likely either not wired correctly or there's a mistake in your set−up. Some terminals allow setting different values (such as baud rate) for send and receive so the receive could be OK but the send bad.

You should also (at the console) try "stty < /dev/ttyS1" (if you use ttyS1) to see that it's set up correctly. It will often be in raw mode (and this is probably OK) with −icanon and −echo etc. If the terminal incorrectly set at half−duplex (HDX), then one set of the characters you see when you type are coming from the terminal itself. If the characters are doubled, then the echos from the computer are OK and you may switch to full−duplex to fix this. But if half−duplex is set and you only see what looks like normal "echos", then they are not coming from the computer as they should be.

If you get a message saying something like "login failed" then if there is no error in typing or in the password, there may be some restrictions on logins which will not allow you to log in. Unfortunately, this message, may not tell you why it failed. See Login Restrictions

18.9 Garbled Login Prompt

This may be due to using the wrong character set, transmission errors due to too high of a baud rate, incompatible baud rates, incompatible parity, or the wrong number of bits per byte. If it's a variety of strange characters you have the wrong character set or a high order bit is being set by mistake. If words are misspelled, try a lower baud rate. For baud, parity, or bits/character incompatibilities you see a lot of the same "error character" which represents a real character that can't be displayed correctly due to an error in parity or baud rate.

If you are using agetty (often just named getty), the agetty program will detect and set parity and/or bits/character if you type something. Try it with a return to see if it fixes it.

18.10 No Login Prompt

If there's no login prompt displayed after hitting the return−key a few times then check the following: Use the "top" or "ps" programs to make sure that a getty (or login) process is actually running on the terminal. Is the terminal getting power? Is the null modem cable plugged in to the correct ports on both the terminal and computer?

If getty isn't running, carefully check the format for calling getty in /etc/inittab. Make sure that it includes the current runlevel (shown by typing the command "runlevel") and that it is valid for your flavor of getty. Sometimes killing getty (it will restart automatically) helps.

Terminal was working OK

Although hardware failures are possible, the problem is likely due to something that someone did by mistake. Did someone do something that might have loosened a cable? Did someone modify /etc/inittab or make some other change to the software so as to prevent terminal login? If this doesn't reveal the cause, continue reading.

More diagnose

The above should solve most cases (unless you've just installed a terminal). Other causes include defective hardware or cables (must be null−modem), terminal setup at wrong baud−rate, terminal in local mode, etc. At this point two different avenues of approach are (you may pursue more than one at a time).

- Diagnose problem from the console
- Measure voltages

Diagnose problem from the console

One test is to try to copy a something to the terminal (It might be a good idea to try this near the start of the installation process before setting up getty). You may use the Linux copy command such as:"cp file_name

/dev/ttyS1". Or your could use: "echo any_word > /dev/ttySx". If it doesn't work, use stty to make the interface as simple as possible with everything disabled (such as hardware flow control: −crtscts; parity, and modem control signals: clocal). Be sure the baud rates and the bits/byte are the same. If nothing happens verify that the port is alive with a voltmeter per the next section.

Measure voltages

If you have a voltmeter handy check for a negative voltage (−4v to −15v) at pin 3 (receive data) at the terminal side of the null−modem cable. The positive lead of the meter should be connected to a good ground (the metal connectors on the ends of cables are often not grounded). If there is no such negative voltage then check for it at the transmit pin (TxD) on the computer (see DB9−DB25 for the pin−out). If it's present there but not at the receive pin (RxD) at the terminal, then the cable is bad (loose connection, broken wire, or not a null−modem). If voltage is absent at the computer, then its serial port is dead. Test it with a software diagnostic test or replace it.

If the serial port is alive, you may want to send a file to it (with modem controls disabled) and watch the signal on a voltmeter (or other electronic gadget). To check for a transmitted signal with a voltmeter, check for a steady negative voltage when the line is idle. Then start sending a file (or start getty). On an analog meter you should see the needle dropping to almost 0 and fluttering about 0 as it measures short−run averages of the bit stream. On a digital meter you will not see the fluctuations as well but you can switch to the AC scale to see the AC voltage created by the flow of bits. If your meter fails to block out DC on the AC scale (the default of most analog meters), then you could get a false high AC reading when looking at the idle DC of −12 V on the AC scale. Without a meter, you could connect a known good device (such as another terminal or an external modem) to the serial port and see if it works OK.

18.11 Types Garbage

If what you see on the screen is not what you type but looks like a foreign language, math symbols, line−drawing character, etc. then it could be that the many of bytes that are echoed to your terminal have the high order bit set (when it shouldn't be). You are looking at the character set (or part of a character set) which has high order bit set. This may happen if you have the baud rate set wrong. Try stty −F /dev/ttyS? from another terminal to check if the baud rate is correct.

Perhaps the wrong character set (font) has been installed. An erroneous escape sequence sent to the terminal could have switched character sets. If you are using the mapchan program to change the keyboard mapping, it could be incorrect.

18.12 Slow: pauses of several seconds between bursts of characters

You likely have mis−set interrupts> See the Serial−HOWTO section starting with "Slow:"

18.13 Terminal doesn't scroll

One reason may be that something is wrong with terminfo. Another reason could be that you are outside the scrolling region set for the terminal. Some stupid programs just assume that your terminal has 24 lines and set the scrolling region for 24 lines (by an escape sequence sent to the terminal) without consulting terminfo to see how many lines there actually are. Then when you use another program it may leave the cursor on line 25 where it becomes trapped and the terminal will not scroll. To avoid this problem, create an environment variable "export LINES=25" and also "stty −F /dev/ttySx rows 25". Then the programs that assume 24 lines will hopefully use 25 lines set the scrolling region accordingly.

18.14 Serial Monitoring/Diagnostics

A few Linux programs will monitor the modem control lines and indicate if they are positive (1) or negative (0).

- statserial (in Debian distribution)
- serialmon (doesn't monitor RTS, CTS, DSR but logs other functions)
- modemstat (only works on Linux PC consoles. Will coexist with the command line)

You may already have them. If not, go to Serial Software. When using these, bear in mind that what you see is the state of the lines at the host computer. The situation at the terminal will be different since some wires are often missing from cables while other wires cross over. As of June 1998, I know of no diagnostic program in Linux for the serial port.

18.15 Local Mode

In local mode, the terminal disconnects from the computer and behaves like a typewriter (only it doesn't type on paper but on the screen). Going back into on−line mode reconnects to the computer allowing you to resume activities at the same point where you left off when you went into "local". This is useful both for testing the terminal and for educational purposes. For some terminals there is no "local mode" but "block mode" may substitute for it. If there is no "block mode", "half duplex" mode might work something like this except that what you type gets sent to the computer also. In this case the computer may echo the characters sent to it resulting in two characters displayed on the screen for every character you type. To prevent this you could shut down the computer, disconnect the RS−232 cable, etc.

When in local mode you may type escape sequences (starting with the ESC key) and observe what they do. If the terminal doesn't work correctly in local mode, it's unlikely that it will work correctly when connected to the computer. If you're not exactly sure what an escape sequence does, you can try it out in local mode. You might also use it for trying out a terminal that is for sale. To get into local mode on some terminals you first enter set−up mode and then select "local" from a menu (or press a certain key). See Getting Into Set−Up (Configuration) Mode.

18.16 Serial Electrical Test Equipment

Breakout gadgets, etc.

While a multimeter (used as a voltmeter) may be all that you need for just a few terminals, simple special test equipment has been made for testing serial port lines. Some are called "breakout ... " where breakout means to break out conductors from a cable. These gadgets have a couple of connectors on them and insert into the serial cable. Some have test points for connecting a voltmeter. Others have LED lamps which light when certain modem control lines are asserted (turned on). Still others have jumpers so that you can connect any wire to any wire. Some have switches.

Radio Shack sells (in 1998) a "RS−232 Troubleshooter" or "RS−232 Line Tester" which checks TD, RD, CD, RTS, CTS, DTR, and DSR. A green light means on (+12 v) while red means off (−12 v). They also sell a "RS−232 Serial Jumper Box" which permits connecting the pins anyway you choose.

Measuring voltages

Any voltmeter or multimeter, even the cheapest that sells for about $10, should work fine. Trying to use other methods for checking voltage is tricky. Don't use a LED unless it has a series resistor to reduce the voltage across the LED. A 470 ohm resistor is used for a 20 ma LED (but not all LED's are 20 ma). The LED will only light for a certain polarity so you may test for + or − voltages. Does anyone make such a gadget for automotive circuit testing?? Logic probes may be damaged if you try to use them since the TTL voltages for which they are designed are only 5 volts. Trying to use a 12 V incandescent light bulb is not a good idea. It won't show polarity and due to limited output current of the UART it probably will not even light up.

To measure voltage on a female connector you may plug in a bent paper clip into the desired opening. The paper clip's diameter should be no larger than the pins so that it doesn't damage the contact. Clip an alligator clip (or the like) to the paper clip to connect up.

Taste voltage

As a last resort, if you have no test equipment and are willing to risk getting shocked (or even electrocuted) you can always taste the voltage. Before touching one of the test leads with your tongue, test them to make sure that there is no high voltage on them. Touch both leads (at the same time) to one hand to see if they shock you. Then if no shock, wet the skin contact points by licking and repeat. If this test gives you a shock, you certainly don't want to use your tongue.

For the test for 12 V, Lick a finger and hold one test lead in it. Put the other test lead on your tongue. If the lead on your tongue is positive, there will be a noticeable taste. You might try this with flashlight batteries first so you will know what taste to expect.

---

19. Repair & Diagnose

Repairing a terminal has much in common with repairing a monitor and/or keyboard. Sometimes the built−in diagnostics of the terminal will tell you what is wrong on the screen. If not, then by the symptoms, one may often isolate the trouble to one of the following: bad keyboard, CRT dead, terminal digital electronics failure. It's best to have a service manual, but even if you don't have one, many terminals may still be repaired.

19.1 Repair Books & Websites

Books

Bigelow, Stephen J.: Troubleshooting & Repairing Computer Monitors, 2nd edition, McGraw−Hill, 1997. Doesn't cover the character generation electronics nor the keyboard.

Websites

The FAQ http://www.repairfaq.org for the newsgroup: sci.electronics.repair is long and comprehensive, although it doesn't cover terminals per se. See the section "Computer and Video Monitors". Much of this information is applicable to terminals as are the sections: "Testing Capacitors", "Testing Flyback Transformers", etc. Perhaps in the future, the "info" on repair in this HOWTO will consist mainly of links to the above FAQ (or the like). Shuford's repair archive of newsgroup postings on terminal repair is another source of info.

19.2 Safety

CRT's use high voltage of up to 30,000 volts for color (less for monochrome). Be careful not to touch this voltage if the set is on and the cover off. It probably won't kill you even if you do since the amount of current it can supply is limited. But it is likely to badly burn and shock you, etc. High voltage can jump across air gaps and go thru cracked insulation so keep your hands a safe distance from it. You should notice the well−insulated high voltage cable connected to one side of the picture tube. Even when the set is off, there is still enough residual voltage on the picture tube cable connection to give you quite a shock. To discharge this voltage when the set is unplugged use a screwdriver (insulated handle) with the metal blade grounded to the picture tube ground cable with a jumper wire. Don't use chassis ground.

The lower voltages (of hundreds of volts) can be even more dangerous since they are not current limited. It is even more dangerous if your hands are wet or if you are wearing a metal watchband, ring or the like. In rare cases people have been killed by it so be careful. The lowest voltages of only several volts on digital circuitry are fairly safe but don't touch anything (except with a well insulated tool) unless you know for sure.

19.3 Appearance of Display

If the display is too dim, turn up the brightness and/or contrast. using knobs on the exterior of the unit (if they exist). If the width, height or centering is incorrect, there are often control knobs for these. For some older terminals one must press an arrow key (or the like) in set−up mode.

You may need to remove the cover to make adjustments, especially on older models. You could arrange things so that a large mirror is in front of the terminal so as to view the display in the mirror while making adjustments. The adjustments to turn may be on a printed circuit board. While a screwdriver (possibly Phillips−head) may be all that's needed, inductors may require special TV alignment tools (plastic hex wrenches, etc.). The abbreviated name of the adjustment should be printed on the circuit board. For example, here are some such names:

- V−Size adjusts the Vertical height (Size)
- H−Size adjusts the Horizontal width (Size). It may be an inductor.
- V−Pos adjusts the Vertical Position
- H−Pos adjusts the Horizontal Position
- V−Lin adjusts Vertical Linearity (Use if width of scan lines differs at the top and bottom of the screen)
- V−Hold adjusts Vertical Hold (Use if screen is uncontrollable scrolling)
- Bright adjusts brightness (an external knob may also exist)
- Sub−Bright adjusts brightness of subdued intensity mode (often the normal mode: dimmer than bold or bright mode).

Changing linearity may change the size so that it will need to be readjusted. A terminal that has been stored for some time may have a small display rectangle on the screen surrounded by a large black border. If it's difficult to adjust, wait a while before adjusting it since it will likely recover some with use (the black borders will shrink).

19.4 Diagnose

Terminal Made a Noise

If the terminal made some noise just before it failed (or when you turn it on after it failed) that noise is a clue to what is wrong. If you hear a sparking noise or see/smell smoke, immediately turn the terminal off to prevent further damage. The problem is likely in the high voltage power supply of several thousand volts. Remove the cover and if the bad spot is not evident, turn it on again for a short time in a dimly lit room to look for arcing. The high voltage cable (runs between the flyback transformer and the side of the picture tube) may have broken insulation that arcs to ground. Fix it with high−voltage insulating dope, or special electrical tape designed say for 10,000 volts.

The flyback transformer (high voltage) may make only a faint clicking or sparking noise if it fails. You may not hear it until you turn the terminal off for a while to rest and then turn it back on again. To track down the noise you may use a piece of small rubber tubing (such as used in automobiles) as a stethoscope to listen to it. But while you are listening for the noise, the terminal is suffering more damage so try find it fast (but not so fast as to risk getting shocked).

A short in the power supply may cause a fuse to blow with a pop. Replacing a blown fuse may not solve the problem as the same short may blow the fuse again. Inspect for any darkened spots due to high heat and test those components. Shorted power transistors may cause the fuse to blow. They may be tested with a transistor checker or even with an ohm−meter. Use the low ohm scale on an ohm−meter so that the voltage applied by the meter is low. This will reduce the possible damage to good components caused by this test voltage.

If the terminal has been exposed to dampness such as being stored in a damp place or near a kitchen with steam from cooking, a fix may be to dry out the unit. Heating a "failed" flyback transformer with a blow dryer for several minutes may restore it.

Terminal Made No Noise

A blank screen may be due to someone turning the brightness control to the lowest level or to aging. The next thing to do is to check the cables for loose or broken connections. If there is no sign of power, substitute a new power cord after making sure that the power outlet on the wall is "hot".

If the keyboard is suspected, try it on another terminal of the same type or substitute a good keyboard. Wiggle the keyboard cable ends and the plug. Wires inside cables may break, especially near their ends. If the break is verified by wiggling it (having the problem go on and off in synchronization with the wiggles), then one may either get a new cable or cut into the cable and re−solder the breaks, etc.

One of the first things to do if the keyboard works is to put the terminal into Local Mode. If it works OK in local, then the problem is likely in the connection to the host computer (or incorrect interface) or in the UART chips of the terminal.

By carefully inspecting the circuitry, one may often find the cause of the problem. Look for discoloration, cracks, etc. An intermittent problem may sometimes be found by tapping on components with a ball−point pen (not the metal tip of course). A break in the conductor of a printed circuit board may sometimes be revealed by flexing the board. Solder that looks like it formed a drop or a solder joint with little solder may need re−soldering. Soldering may heat up transistors (and other components) and damage them so use a heat sink if feasible.

If you have a common brand of terminal, you may be able to search newsgroup postings on the Internet to find out what the most frequent types of problems are for your terminal and perhaps information on how to fix them.

To see if the digital electronics work, try (using a good keyboard) typing at the bad terminal. Try to read this typing at a good terminal (or the console) using the copy command or with a terminal communication program such as minicom. You may need to hit the return key at the terminal in order to send a line. One may ask the bad terminal for its identity etc. from another terminal. This will show if two−way communication works.

19.5 Error Messages on the Screen

You are in luck if you see an error message on the screen. This usually happens when you first turn the terminal on.

Keyboard Error

This usually means that the keyboard is not plugged in, or that the connection is loose. For more serious problems see Keyboards

Checksum Error in NVR

NVR is "Non−Volatile RAM". This means that the NVR where the set−up information is stored has become corrupted. The terminal will likely still work but the configuration that was last saved when someone last configured the terminal has likely been lost. Try configuring again and then save it. It might work. On very old terminals (early 1980's) there was a battery−powered CMOS to save the configuration so in this case the problem could be just a dead battery. Sometimes the EEPROM chip (no battery needed) goes bad after too many saves. It may be hard to find. If you can't fix it you are either stuck with the default configuration or you may have escape sequences sent to the terminal when you start it up to try to configure it.

19.6 Capacitors

Electrolytic capacitors have a metal shell and are may become weak or fail if they set for years without being used. Sometimes just leaving the terminal on for a while will help partially restore them. If you can, exercise any terminals you have in storage by turning them on for a while every year or two.

19.7 Keyboards

Interchangeability

The keyboards for terminals are not the same as keyboards for PC's. The difference is not only in the key layout but in the codes generated when a key is pressed. Also, keyboards for various brands and models of terminals are not always interchangeable with each other. Sometimes one get an "incompatible" keyboard to partially work on a terminal. All the ASCII keys will work OK, but special keys such as set−up and break will not work correctly.

How They Work

Most keyboards just make a simple contact between two conductors when you press a key. Electronics inside a chip in the keyboard converts this contact closure into a code sent over the keyboard's external cable.

Instead of having a separate wire (or conductor) going from each key to the chip, the following type scheme is used. Number the conductors say from 1−10 and A−J. For example: conductor 3 goes to several keys and conductor B goes to several keys, but only one key has both conductors 3 and B going to it. When that key is pressed, a short circuit is established between 3 and B. The chip senses this short and knows what key has been pressed. Such a scheme reduces the number of conductors needed (and reduces the number of pins needed on the chip). It's a similar scheme to what is called a "crossbar switch".

Modern vs Old Keyboards

While the modern keyboard and the old fashioned type look about the same, the mechanics of operation are different. The old ones have individual key switches under the key−caps with each switch enclosed in a hard plastic case. The modern ones use large flexible plastic sheets (membrane) the size of the keyboard. A plastic sheet with holes in it is sandwiched between two other plastic sheets containing printed circuits (including contact points). When you press a key, the two "printed" sheets are pressed together at a certain point, closing the contacts printed on the sheets at that point.

One Press Types 2 Different Characters

If, due to a defect, conductors 3 and 4 become shorted together then pressing the 3−B key will also short 4 and B and the chip will think that both keys 3−B and 4−B have been pressed. This is likely to type 2 different characters when all you wanted was one character.

Keyboard doesn't work at all

If none of the keys work try another keyboard (if you have one) to verify that the keyboard is the problem. One cause is a broken wire inside the cord (cable) that connects it to the terminal. The most likely location of the break is at either end of the cord. Try wigging the ends of the cord while tapping on a key to see if it works intermittently. If you find a bad spot, you may carefully cut into the cord with a knife at the bad spot and splice the broken conductor. Sometimes just a drop of solder will splice it. Seal up the cord with electrical tape, glue, or caulk. A keyboard that has gotten wet may not work at all until it's dry.

Typing b displays bb, etc. (doubled)

If all characters appear double there is likely nothing wrong with the keyboard. Instead, your terminal has likely been incorrectly set up for half−duplex (HDX or local echo=on) and every character you type is echoed back both from the electronics inside your terminal and from your host computer. If the two characters are not the same, there may be a short circuit inside your keyboard. See One Press Types 2 Different Characters

Row upon row of the same character appears

This may happen when auto−repeat is enabled and a key is held pressed down (or the like). It may be a key that sticks down when typed or it could be an electrical short that has the same effect.

Key sticks in down position (individual switches)

If it's a stuck key on a keyboard with individual switches, a good way to fix it is to remove the keycap (if it's removable). See id="kbd_sw" name="Keyboards with individual switches">. Use a small amount of cleaner on the push rod. Some keys stick due to stickiness on the keycap bottom surface (and where it hits on the switch). If the key sticks in the fully down position this could be the problem. So clean this this area too.

Hitting the key a lot to exercise it may help, but the problem is likely to return. If you suspect the push rod is sticking you might try to type it while pushing sideways on it with a small screwdriver. You should push it sideways in one of the four directions and try different directions. What you are doing by this is attempting to force out a foreign particle that is rubbing on the side of the key's push−rod and making it stick. Again, the problem may return later.

To test the key, push it down very slowly and see if it sticks. Also push it sideways a little as you're pushing it down. If you hit it fast or push it straight down, then you may not observe the stickiness.

Key electrically shorted

If you suspect that a key is shorted out fix it by cleaning the contacts per Cleaning Keyboard Contacts. If this problem happens at the login prompt see Key shorted.

Liquid spilled on the keyboard

If water or watery liquid has been spilled on the keyboard (or if it was exposed to rain, heavy dew, or dampness) some (or all) keys may not work right. The dampness may cause a key to short out (like it was pressed down all the time) and you may see the screen fill up with that letter if auto−repeat is enabled. If it's gotten wet and then partially (or fully) dried out, certain keys may not work due to deposits on the contact surfaces. For the modern type of keyboard, one may readily take apart the plastic sheets inside and dry/clean them. For the old type one may let it dry out in the sun or oven (low temp.). When it's dry it may still need contact cleaner on some keys as explained below.

Cleaning keyboard contacts

Keyboards with membranes

On some newer keyboards, the plastic sheets (membranes) are easy to remove for inspection and cleaning if needed. You only need to remove several screws to take apart the keyboard and get to the sheets. On some old IBM keyboards the sheets can't be removed without breaking off many plastic tabs which will need to be repaired with glue to put back (probably not worthwhile to repair). Such a keyboard may sometimes be made to work by flexing, twisting, and/or pounding the assembly containing the plastic sheets.

Keyboards with individual switches

What follows is for older keyboards that have separate hard plastic switches for each key. Before going to all the work of cleaning electrical contacts first try turning the keyboard upside−down and working the bad keys. This may help dislodge dirt, especially if you press the key hard and fast to set up vibration. Pressing the key down and wiggling it from side to side, etc. often helps.

Often the key−caps may be removed by prying them upward using a small screwdriver as a lever while preventing excessive tilting with a finger. There exists a special tool known as keycap puller but you can get by without it. (Warning: Key−caps on modern keyboards don't pry up.) The key−cap may tilt a bit and wobble as it comes loose. It may even fly up and onto the floor. Then you have two choices on how to clean the contacts: Use contact cleaner spray directly on top of the key switch, or take the key switch apart and clean it. Still another choice is to replace the key switch with a new or used one.

Directly spraying contact cleaner or the like (obtained at an electronics store) into the top of the key switch is the fastest method but it may not work. Before spraying, clean the area around it a little. With the keyboard live (or with the key contacts connected to an ohm−meter) use the tube which came with the spray to squirt cleaner so it will get inside the key switch. Don't let the cleaning liquid get under nearby keys where it may pick up dust and then seep (with the dust) into adjacent key switches. If you make this mistake you may fix one key but damage nearby keys. If this should happen, immediately work (repeatedly press) the affected nearby keys until they continue to work OK.

You might tilt the keyboard so that the cleaner flows better into the contacts. For the CIT101e terminal with an Alps keyboard, this means tilting the digit row up toward the ceiling. Work the key switch up and down with a pen or small screwdriver handle to avoid getting the toxic cleaner liquid on your skin (or wear gloves). You might try turning the keyboard upside−down while working the key to drain off remaining cleaner. I don't usually do this. The more cleaner you squirt in the more likely it will fix it but it is also more likely to do more damage to the plastic or contaminate adjacent keys, so use what you think is just enough to do the job. Once the key works OK, work it up and down a little more and test it a half minute later, etc. to make sure it will still work OK.

Sometimes a key works fine when the contacts inside are saturated with contact cleaner liquid. But when the liquid dries a few minutes later then the resulting scale deposit left from the evaporation of the cleaning liquid on the contacts, prevents good contact. Then the key may work erratically (if at all). Operating the key when the liquid is drying inside may help. Some switches have the contacts nearly sealed inside so little if any contact cleaner reaches the contacts. The cleaner that does get to the contacts may carry contamination with it (cleaning around the tops before spraying helps minimize this).

If you need to disassemble the key switch, first inspect it to see how it is installed and comes apart. Sometimes one may remove the cover of the switch without removing the switch from the keyboard. To do this pry up (or pull up) the top of the key switch after prying apart thin plastic tabs that retain it. Don't pry too hard or you may break the thin plastic. If this can't be done, you may have to unsolder the switch and remove it in order to take it apart (or replace it). Once the switch has been taken apart you still may not be able to see the contacts if the contact surfaces are sandwiched together (nearly touching). You may get contact cleaner on the contacts by slightly prying apart the conducting surfaces and squirting cleaner between them. There may be some kind of clip holding the contact surfaces together which needs to be removed before prying these surfaces apart. With cleaner on the contacts, work them. Tilting the keyboard or inverting it may help. Take care not to loose small parts as they may fly up into the air when taking apart a key switch.

---

20. Appendix A: General

20.1 List of Linux Terminal Commands

Sending a command to the terminal

- setterm: long options
- tput: terse options
- tset: initializes only
- clear: clears screen
- setterm −reset: sends reset string

Configuring the terminal device driver

- Setserial:
- Stty

Terminfo

- Terminfo Compiler (tic) terminfo compiler & translator
- toe: shows list of terminals for which you have terminfo files
- infocmp compares or displays terminfo entries

Other

- gitkeys: shows what bytes each key sends to the host.
- tty: shows what tty port you are connected to.
- set (or tset −q): shows the value of TERM, the terminfo entry name
- tset: sets TERM interactively and initializes

20.2 The Internet and Books

Terminal Info on the Internet

- Shuford's Website at the University of Tennessee has a great deal of useful information about text terminals.
- VT terminal information and history http://www.vt100.net/
- Boundless purchased the VT and Dorio terminal business from DEC. To get Specs select either ADDS, VT, or DORIO links. Then select a "data sheet" link. Then on the data sheet select the "Go to Specs" link.
- Wyse is a major manufacturer of terminals. Wyse text−terminals database For current models see Wyse terminals.
- Teemworld Escape Sequences is a list of escape sequences (and control codes) for some terminal emulations (including VT 100, 300, 420, and Wyse).
- ncurses FAQ
- comp.terminals is the newsgroup for terminals

Books related to terminals

- EIA−232 serial port see EIA−232 (RS−232) Books.
- Repair see Repair Books & Websites.
- Terminfo database see Termcap Documents

Entire books on terminals

As far as I know, there is no satisfactory book on text terminals Although this HOWTO has been published as a book, I don't suggest that that you buy it if you have access to the online version which I'm improving on every month or so. The following are mainly of historical interest:

- Handbook of Interactive Computer Terminals by Duane E. Sharp; Reston Publishing Co. 1977. (mostly obsolete)

* Communicating with Display Terminals by Roger K. deBry; McGraw−Hill 1985. (mostly on IBM synchronous terminals)

The "HANDBOOK ... " presents brief specifications of over 100 different models of antique terminals made in the early 1970's by over 60 different companies. It also explains how they work physically but incorrectly shows a diagram for a CRT which uses electrostatic deflection of the electron beam (p. 36). Terminals actually used magnetic deflection (even in the 1970's). This book explains a number of advanced technical concepts such as "random scan" and "color penetration principle".

The "COMMUNICATING ... " book in contrast to the "Handbook ... " ignores the physical and electronic details of terminals. It has an entire chapter explaining binary numbers (which is not needed in a book on terminals since this information is widely available elsewhere). It seems to mostly cover old IBM terminals (mainly the 3270) in block and synchronous modes of operation. It's of little use for the commonly used ANSI terminals used today on Unix−like systems. Although it does discuss them a little it doesn't show the various wiring schemes used to connect them to serial ports.

Books with chapters on terminals

These chapters cover almost nothing about the terminals themselves and their capabilities. Rather, these chapters are mostly about how to set up the computer (and its terminal driver) to work with terminals. Due to the differences of different Unix−like systems, much of the information does not not apply to Linux.

* Unix Power Tools by Jerry Peck et. al. O'Reilly 1998. Ch. 5 Setting Up Your Terminal, Ch. 41: Terminal and Serial Line Settings, Ch. 42: Problems With Terminals
* Advanced Programming in the Unix Environment by W. Richard Stevens Addison−Wesley, 1993. Ch. 11: Terminal I/O, Ch. 19: Pseudo Terminals
* Essential System Administration by Aleen Frisch, 2nd ed. O'Reilly, 1998. Ch. 11: Terminals and Modems.

The "UNIX POWER TOOLS" book has 3 short chapters on text terminals. It covers less ground than this HOWTO but gives more examples to help you.

The "ADVANCED PROGRAMMING ... " Chapter 11 covers only the device driver included in the operating system to deal with terminals. It explains the parameters one gives to the stty command to configure the terminal.

The "ESSENTIAL SYSTEM ..." book's chapter has more about terminals than modems. It seems well written.

20.3 Non−Linux OS'

Under Microsoft's DOS one may use the DOS command ctty so that the DOS command line will display on a serial terminal. Unfortunately, one can then no longer use the computer monitor since DOS is not a multiuser operating system.

For other unix−like OSs, the configuration of the host computer for terminals is usually significantly different than for Linux. Here are some links to on−line manuals for unix−like systems.

* SCO's OpenServer Adding Serial Terminals in SCO OpenServer Handbook.
* Hewlett−Packard's HP−UX Configuring Terminals and Modems

21. Appendix B: Escape Sequence Commands Terminology

These are sometimes called "control sequences". This section of Text−Terminal−HOWTO is incomplete (and may never be complete as there are such a huge number of control sequences). This section is for reference and perhaps really belongs in something that would be called "Text−Terminal−Programming−HOWTO".

An example of an ANSI standard escape sequence is ESC[5B which moves the cursor down 5 lines. ESC is the Escape character. The parameter 5 is included in the sequence. If it were 7 the cursor would move down 7 lines, etc. A listing for this sequence as "move cursor down x lines: ESC[xB" is easy to to understand. But command jargon such as: "tertiary device attribute request" is less comprehensible. This section will try to explain some of the more arcane jargon used for escape sequence commands. A full listing (including the escape sequence codes for the ANSI standard) is a "wish list" project. Since many escape sequences do the same thing as is done when setting up the terminal with Set−Up Options, such escape sequences options will not be repeated here.

21.1 Esc Sequence List

For a list of many (but not all) escape sequences for various terminals see Teemworld Escape Sequences. These are used for terminal emulation and are not always the same as on the corresponding real terminal. A list for VT (not maintained) may be found at Emulators FAQ. Search for "VT". For downloading manuals see VT Manuals.

21.2 8−bit Control Codes

Table of 8−bit DEC control codes (in hexadecimal). Work on VT2xx or later. CSI is the most common.

```
ACRONYM  FULL_NAME                          HEX     REPLACES
IND Index (down one line)                   84      ESC D
NEL Next Line                               85      ESC E
RI  Reverse Index (one line up)             8D      ESC M
SS2 Single Shift 2                          8E      ESC N
SS3 Single Shift 3                          8F      ESC O
DCS Device Control String                   90      ESC P
CSI Control Sequence Introducer)            9B      ESC [
ST  String Terminator                       9C      ESC \
```

21.3 Printer Esc

- Auto Print on/off: When on, data from the host is also teed (sent) to the printer port of the terminal (and also shows on the terminal screen).
- Print Controller on/off: When on, data from the host is sent only to the printer (nothing shows on the terminal screen).

21.4 Reports

These sequences are usually a request sent from the host to request a report from the terminal. The terminal responds by sending a report (actually another escape sequence) to the host which has embedded in it certain values telling the host about the current state of the terminal. In some cases a report may be sent to the host even if it wasn't asked for. This sometimes happens when set−up is exited. By default no unsolicited reports should be sent.

- Request for Status (Report Operating Status): Meaning of replies for VT100 is either "I'm OK" or "I'm not OK"
- Request for Device Attributes: The "device" is usually the printer. Is there a printer? Is it ready?
- Reqest for Tertiary Device Attributes (VT): Reply is report that was entered during set−up. The tertiary device is the 3rd device (the printer or auxiliary port device ??). The 1st device may be the host computer and the 2nd device the terminal.
- Request for Terminal Parameters: What is the parity, baud rate, byte width, etc. This request doesn't seem to make much sense, since if the host didn't already know this it couldn't communicate with the terminal or send a reply.

21.5 Cursor Movements

The cursor is where the next character received from the host will be displayed. Most of the cursor movements are self−explanatory. "index cursor" means to move the cursor down one line. Cursor movements may be relative to the current position such as "move 4 spaces left" or absolute such as "move to row 3, column 39". Absolute is called "Direct Cursor Positioning" or "Direct Cursor Addressing".

The home position is row 1 col. 1 (index origin is 1). But where this home position is on the physical screen is not completely clear. If "Cursor Origin Mode" = "Relative Origin Mode" is set, then home is at the top of the scrolling region (not necessarily the top of the screen) at the left edge of the screen. If "Absolute Origin Mode" is set (the same as unsetting any of the two modes in the previous sentence) then home is at the upper left corner of the screen. On some old terminals if "Cursor Origin Mode" is set it means that it's relative.

21.6 Pages (definition)

See Pages for an explanation of pages. There are a number of escape sequences to deal with pages. Text may be copied from one page to another and one may move the cursor from page to page. Switching pages may or may not be automatic: when the screen becomes full (page 1) then more data from the host goes to page 2. The cursor may only be on one page at a time and characters which are sent to the terminal go there. If that page is not being displayed, new text will be received by the terminal and go into display memory, but you will not see it (until the terminal is switched to that page).

---

22. Appendix C: Serial Communications on EIA−232 (RS−232)

22.1 Intro to Serial Communication

(Much of this section is now found in Serial−HOWTO.) Text terminals on Unix−like systems (and on PC's) are usually connected to an asynchronous 232 serial port of a computer. It's usually a RS−232−C, EIA−232−D, or EIA−232−E. These three are almost the same thing. The original RS prefix became EIA (Electronics Industries Association) and later EIA/TIA after EIA merged with TIA (Telecommunications Industries Association). The EIA−232 spec provides also for synchronous (sync) communication but the hardware to support sync is almost always missing on PC's. The RS designation is obsolete but is still in use. EIA will be used in this article.

The serial port is more than just a physical connector on the back of a computer or terminal. It includes the associated electronics which must produce signals conforming to the EIA−232 specification. The standard connector has 25 pins, most of which are unused. An alternative connector has only 9 pins. One pin is used to send out data bytes and another to receive data bytes. Another pin is a common signal ground. The other "useful" pins are used mainly for signalling purposes with a steady negative voltage meaning "off" and a steady positive voltage meaning "on".

The UART (Universal Asynchronous Receiver−Transmitter) chip does most of the work. Today, the functionality of this chip is usually built into another chip.

22.2 Voltages

Voltage for a bit

At the EIA−232 serial port, voltages are bipolar (positive or negative with respect to ground) and should be about 12 volts in magnitude (some are 5 or 10 volts). For the transmit and receive pins +12 volts is a 0−bit (sometimes called "space") and −12 volts is a 1−bit (sometimes called "mark"). This is known as inverted logic since normally a 0−bit is both false and negative while a one is normally both true and positive. Although the receive and transmit pins are inverted logic, other pins (modem control lines) are normal logic with a positive voltage being true (or "on" or "asserted") and a negative voltage being false (or "off" or "negated"). Zero voltage has no meaning (except it usually means that the PC is powered off).

A range of voltages is allowed. The specs say the magnitude of a transmitted signal should be between 5 and 15 volts but must never exceed 25 V. Any voltage received under 3 V is undefined (but some terminals will accept a lower voltage as valid). One sometimes sees erroneous claims that the voltage is commonly 5 volts (or even 3 volts) but it's usually 11−12 volts. If you are using a EIA−422 port on a Mac computer as an EIA−232 (requires a special cable) or EIA−423 then the voltage will actually be only 5 V. The discussion here assumes 12 V. There is much confusion about voltages on the Internet.

Note that normal computer logic normally is just a few volts (5 volts was once the standard) so that if you try to use test equipment designed for testing 3−5 volt computer logic (TTL) on the 12 volts of a serial port, it may damage the test equipment.

Voltage sequence for a byte

The transmit pin (TxD) is held at −12 V (mark) at idle when nothing is being sent. To start a byte it jumps to +12 V (space) for the start bit and remains at +12 V for the duration (period) of the start bit. Next comes the low−order bit of the data byte. If it's a 0−bit nothing changes and the line remains at +12 V for another bit−period. Then comes the next bit, etc. Finally, a parity bit may be sent and then a −12 V (mark) stop bit. The line remains at −12 V (idle) until the next start bit. Note that there is no return to 0 volts and thus there is no simple way (except by a synchronizing signal) to tell where one bit ends and the next one begins for the case where 2 consecutive bits are the same polarity (both zero or both one).

A 2nd stop bit would also be −12 V, just the same as the first stop bit. Since there is no signal to mark the boundaries between these bits, the only effect of the 2nd stop bit is that the line must remain at −12 V idle twice as long. The receiver has no way of detecting the difference between a 2nd stop bit and a longer idle time between bytes. Thus communications works OK if one end uses one stop bit and the other end uses 2 stop bits, but using only one stop bit is obviously faster. In rare cases 1 1/2 stop bits are used. This means that the line is kept at −12 V for 1 1/2 time periods (like a stop bit 50% wider than normal).

22.3 Parity Explained

Characters are normally transmitted with either 7 or 8 bits (of data). An additional parity bit may (or may not) be appended to this resulting in a byte length of 7, 8 or 9 bits. Some terminal emulators and older terminals do not allow 9 bits. Some prohibit 9 bits if 2 stop bits are used (since this would make the total number of bits too large: 12 bits total).

The parity may be set to odd, even or none (mark and space parity may be options on some terminals). With odd parity, the parity bit is selected so that the number of 1−bits in a byte, including the parity bit, is odd. If a such a byte gets corrupted by a bit being flipped, the result is an illegal byte of even parity. This error will be detected and if it's an incoming byte to the terminal an error−character symbol will appear on the screen. Even parity works in a similar manner with all legal bytes (including the parity bit) having an even number of 1−bits. During set−up, the number of bits per character usually means only the number of data bits per byte (7 for true ASCII and 8 for various ISO character sets).

A "mark" is a 1−bit (or logic 1) and a "space" is a 0−bit (or logic 0). For mark parity, the parity bit is always a one−bit. For space parity it's always a zero−bit. Mark or space parity only wastes bandwidth and should be avoided when feasible. "No parity" means that no parity bit is added. For terminals that don't permit 9 bit bytes, "no parity" must be selected when using 8 bit character sets since there is no room for a parity bit.

22.4 Forming a Byte (Framing)

In serial transmission of bytes via EIA−232 ports, the low−order bit is always sent first. Serial ports on PC's use asynchronous communication where there is a start bit and a stop bit to mark the beginning and end of a byte. This is called framing and the framed byte is sometimes called a frame. As a result a total of 9, 10, or 11 bits are sent per byte with 10 being the most common. 8−N−1 means 8 data bits, No parity, 1 stop bit. This adds up to 10 bits total when one counts the start bit. One stop bit is almost universally used. At 110 bits/sec (and sometimes at 300 bits/sec) 2 stop bits were once used but today the 2nd stop bit is used only in very unusual situations (or by mistake since it seemingly still works OK that way).

22.5 Limitations of EIA−232

Low Speed & Short Distance

The conventional EIA−232 serial port is inherently low speed and is severely limited in distance. Ads often read "high speed" but it can only work at high speed over very short distances such as to a modem located right next to the computer. All of the wires use a common ground return so that twisted−pair technology (needed for high speeds) can't be used without additional hardware. However some computers have more modern interfaces. See Successors to EIA−232.

It is somewhat tragic that the RS−232 standard from 1969 did not use twisted pair technology which could operate about a hundred times faster. Twisted pairs have been used in telephone cables since the late 1800's. In 1888 (over 110 years ago) the "Cable Conference" reported its support of twisted−pair (for telephone systems) and pointed out its advantages. But over 80 years after this approval by the "Cable Conference", RS−232 failed to utilize it. Since RS−232 was originally designed for connecting a terminal to a low speed modem located nearby, the need for high speed and longer distance transmission was apparently not recognized.

Successors to EIA−232

See the Serial−HOWTO section "Other Serial Devices" for a longer discussion about non−EIA−232 ports. A number of EIA standards have been established for higher speeds and longer distances using twisted−pair (balanced) technology. Balanced transmission can sometimes be a hundred times faster than unbalanced EIA−232. For a given speed, the distance (maximum cable length) may be many times longer with twisted pair. Few terminals seem to support them. While many terminals also support EIA−423 is is almost like EIA−232 but is only 5 volts and somewhat higher speeds (without using twisted pair). Twisted pair includes EIA−422, EIA−530−A, HSSI (High Speed Serial Interface), USB (Universal Serial Bus), and of course ethernet.

Line Drivers

For a text terminal, the EIA−232 speeds are fast enough but the usable cable length is often too short. Balanced technology could fix this. The common method of obtaining balanced communication with a text terminal is to install 2@ line drivers in the serial line to convert unbalanced to balanced (and conversely). They are a specialty item and are expensive if purchased new.

22.6 Synchronization & Synchronous

How "Asynchronous" is Synchronized

Per EIA−232 there are only two states of the transmit (or receive) wire: mark (−12 V) or space (+12 V). There is no state of 0 V. Thus a sequence of 1−bits is transmitted by just a steady −12 V with no markers of any kind between bits. For the receiver to detect individual bits it must always have a clock signal which is in synchronization with the transmitter clock. Such clocks generate a "tick" in synchronization with each transmitted (or received) bit.

For asynchronous transmission, synchronization is achieved by framing each byte with a start bit and a stop bit (done by hardware). The receiver listens on the line for a start bit and when it detects one it starts its clock ticking. It uses this clock tick to time the reading of the next 7, 8 or 9 bits. (It actually is a little more complex than this since several samples of a bit are often taken and this requires additional timing ticks.) Then the stop bit is read, the clock stops and the receiver waits for the next start bit. Thus async is actually synchronized during the reception of a single byte but there is no synchronization between one byte and the next byte.

Defining Asynchronous vs Synchronous

Asynchronous (async) means "not synchronous". In practice, an async signal is what the async serial port sends and receives which is a stream of bytes each delimited by a start and stop bit. Synchronous (sync) is most everything else. But this doesn't explain the basic concepts.

In theory, synchronous means that bytes are sent out at a constant rate one after another in step with a clock signal tick. There is often a separate wire or channel for sending the clock signal. Asynchronous bytes may be sent out erratically with various time intervals between bytes (like someone typing characters at a keyboard).

There are borderline situations that need to be classified as either sync or async. The async serial port often sends out bytes in a steady stream which would make this a synchronous case but since they still have the start/stop bits (which makes it possible to send them out erratically) its called async. Another case is where data bytes (without any start−stop bits) are put into packets with possible erratic spacing between one packet and the next. This is called sync since the bytes within each packet must be transmitted synchronously.

Synchronous Communication

Did you ever wonder what all the unused pins are for on a 25−pin connector for the serial port? Most of them are for use in synchronous communication which is seldom implemented on PC's. There are pins for sync timing signals as well as for a sync reverse channel. The EIA−232 spec provides for both sync and async but PC's use a UART (Universal Asynchronous Receiver/Transmitter) chip such as a 16450, 16550A, or 16650 and can't deal with sync. For sync one needs a USART chip or the equivalent where the "S" stands for Synchronous. Since sync is a niche market, a sync serial port is likely to be quite expensive.

Besides the sync part of the EIA−232, there are various other EIA synchronous standards. For EIA−232, 3 pins of the connector are reserved for clock (or timing) signals. Sometimes it's a modem's task to generate some timing signals making it impossible to use synchronous communications without a synchronous modem (or without a device called a "synchronous modem eliminator" which provides the timing signals).

Although few serial ports are sync, synchronous communication does often take place over telephone lines using modems which use V.42 error correction. This strips off the start/stop bits and puts the date bytes in packets resulting in synchronous operation over the phone line.

22.7 Block Mode

Intro to Block Mode

Block mode is seldom used with Linux. In block mode when one types at a terminal, the results are saved in the terminal memory and are not sent just yet to the host computer. Such terminals often have built−in editing capabilities. When the user presses certain keys (such as the send key) what has been saved in the terminal memory is sent to the host computer. Now the Linux editors vi and emacs, react instantly to pressing certain keys but in the above situation such keys will be pressed and nothing will happen since nothing is sent when a key is pressed. Thus using a block mode terminal will not allow the use of such interactive programs. The old IBM mainframe interface uses block mode (see IBM Terminals so many IBM terminals are block−mode only and also synchronous (see Section Synchronization & Synchronous).

Types of Block Modes, Forms

Block mode may itself have various sub−modes such as "page" (a page at a time) and "line" (a line at a time). Some terminals have both block transmission modes and conventional character modes and may be switched from one mode to another. Async terminals which have block modes include HP2622A, VT130, VT131, VT330, VT340, and Visual500. Many later model terminals can emulate block mode. Block modes may include a forms capability where the host computer sends a form to the terminal. Then the user fills it out and hits the send key which sends only the data in the form back to the host computer. The form itself (not the data) is displayed on the screen in protected fields which don't get transmitted to the host.

Efficiency

Block mode takes a great deal of load off the host computer, especially if the host computer's hardware is designed for block modes (as IBM mainframes were). In character mode every character typed is sent immediately to the serial port and usually causes an interrupt at the host computer. The host that receives the byte must stop whatever it is doing and fetch that character from the port hardware. Even with UART's that have FIFO hardware buffers, the hardware timeout is normally only the transmission time of 3 bytes so that an interrupt is usually issued for every character typed.

In true block mode a long block of characters is received using only one interrupt. If block mode is used with conventional async FIFO serial ports, an interrupt is needed only every 14 bytes since they have 16−byte hardware buffers. Thus much of the load and overhead of interrupt handling is eliminated and the computer has more time to do other tasks when block mode is used.

A significant savings for block mode occurs if the terminal is connected to its host via a network. Without block mode, every character (byte) typed is sent in its own packet including all the overhead bytes (40 in a TCP/IP packet as used on the Internet). With block mode, a large number of characters are sent in a single packet.

22.8 EIA−232 (RS−232) Books

(Note: The first book covers much more than just EIA−232.)

- Black, Uyless D.: Physical Layer Interfaces & Protocols, IEEE Computer Society Press, Los Alamitos, CA, 1996.
- Campbell, Joe: The RS−232 Solution, 2nd ed., Sybex, 1982.
- Putnam, Byron W.: RS−232 Simplified, Prentice Hall, 1987.
- Seyer, Martin D.: RS−232 Made Easy, 2nd ed., Prentice Hall, 1991.

22.9 Serial Software

See Serial Software for Linux software for the serial ports including getty and port monitors.

23. Appendix D: Notes by Brand Name

Here are notes by brand name that were too specific to a certain terminal to be put elsewhere in this HOWTO. If you have some info to contribute on a certain terminal that is not covered elsewhere, it could go here. Various models and brands often have much in common which only need be written about in one place. It would be nice to have for each terminal model, a large set of links linking to the documentation relevant to that model (including escape codes). There are so many models of terminals that such a task would be quite onerous and I, David Lawyer (as of 1998), have no intention of attempting this. If terminal manufacturers would only make their manuals available on the net, then all this might not be needed.

23.1 CIT

CIT terminals were made in Japan in the 1980's for CIE Terminals. They ceased to be imported in the late 1980's. The company, CIE, still makes CItoh printers (in 1997) but has no parts for its abandoned terminals. Ernie at (714) 453−9555 in Irvine CA sells (in 1997) some parts for models 224, 326, etc. but has nothing for the 80 and 101. (The document you are now reading was written mostly on the 101e.)

To save the Set−Up parameters press ^S when in Set−Up mode. cit80: Contrast: knob on rear of terminal, cit101e: Brightness: use up/down arrow keys in Set−Up mode.

23.2 IBM Terminals

Don't confuse IBM terminals with IBM PC monitors. Many IBM terminals don't use ASCII but instead use an 8−bit EBCDIC code. It's claimed that in EBCDIC the bit order of transmission is reversed from normal with the high−order bit going first. The IBM mainframe communication standards are a type of synchronous communication in block mode (sends large packets of characters). Two standards are "BISYNC" and "SNA" (which includes networking standards). Many of their terminals connect with coax cable (RG62A/U) and naive persons may think the "BNC" connecter on the terminal is for ethernet (but it's not).

While this IBM system is actually more efficient than what is normally used in Linux, terminals meeting this IBM standard will not currently work with Linux. However, some IBM terminals are asynchronous ASCII terminals and should work with Linux on PC's. The numbers 31xx may work with the exception that 317x and 319x are not ASCII terminals. Before getting an IBM terminal, make sure there is a termcap (terminfo) for it. If their isn't, it likely will not work with Linux. Even if there is a terminfo, it may not work. For example, there is a termcap for 327x but the 3270 is an EBCDIC synchronous terminal.

The 3270 series includes the 3278 (late 1970's), 3279 with color and graphics, and the 3274 terminal controller (something like the 3174). They may be used for both BISYNC and SNA. The 3290 has a split screen (splits into quarters).

The synchronous IBM terminals don't connect directly to the IBM mainframe, but connect to a "terminal controller" (sometimes called "cluster controller" or "communication controller"). Some of these controllers can convert a synchronous signal to asynchronous so that in this case a synchronous terminal could indirectly connect to a Unix−like host computer via its serial port. But there is still a major problem and that is block transmission. See section Block Mode.

IBM 3153

It's claimed that the Aux port is DCE and uses a straight−thru cable.

23.3 Teletypes

These are antiques and represent the oldest terminals. They are like remotely controlled typewriters but are large and noisy. Made by the Teletype Corp., the first models were made in the 1920's and predate the computer by over 30 years. Early models used electro−mechanical relays and rotating distributors instead of electronics. Their Baudot code was only 5−bits per character as compared to 7−bit ASCII. See the book "Small Computer Systems Handbook" by Sol Libes, Hayden Books, 1978: pp. 138−141 ("Teletypes").

23.4 VT (DEC)

Digital Equipment Corporation made the famous VT series of terminals including the commonly emulated VT100. In 1995 they sold their terminal business to SunRiver which is now named Boundless Technologies. Detailed VT terminal information, some manuals, and history is at http://www.vt100.net/. Other information is available at Shuford's Website. Information on current products is available from the Boundless website. See Terminal Info on the Internet.

VT220: Some have a BNC connector for video output (not for input). Sometimes people erroneously think this is for an ethernet connection.

VT520: Supports full DTR/DSR flow control.

23.5 Wyse

For specs on terminals see http://www.wyse.com/service/support/kbase/wyseterm.asp. This will also lead to some FAQ's for terminal numbers under 100 (such as WY60). For the specs on more recent terminals see See Wyse terminals.

Wyse 60

Display adjustments (must remove cover): Brightness VR202, Height VR302, Width VR101 (also affects height). If you want to use it in Native Personality, then the arrow−key codes will conflict with the codes used in vi (such as ^L). To fix this set "Application key mode" with ESC   3. This results in the arrow keys sending 0xd1 − 0xd4. Due to a bug in the readline interface of the Bash shell, you need to edit /etc/inputrc so that the arrow keys will work in Bash. See Bugs in Bash

Wyse 85

Can emulate VT52/VT100/VT200. Press F3 for setup. Scroll thru setup with up/down keys.

Wyse 99−GT

Here is the setup Menus of the Wyse99GT (late 1980's). Note that TERM means "termination" (character) and not "terminal".

```
    WYSE 99-GT Terminal Set-Up as used at the University of CA, Irvine
                  by David Lawyer, April 1990

                          F1 DISP:
  COLUMNS=80              LINES=24                CELL SIZE=10 X 13
  STATUS LINE=STANDARD    BACKGROUND=DARK         SCROLL SPEED=JUMP
  SCREEN SAVER=OFF        CURSOR=BLINK BLOCK      DISPLAY CURSOR=ON
  ATTRIBUTE=CHAR          END OF LINE WRAP=ON     AUTO SCROLL=ON
  ----------------------------------------------------------------------------
                          F2  GENERAL:
  PERSONALITY=VT 100      ENHANCE=ON              FONT LOAD=OFF
  COMM MODE=FULL DUPLEX   RCVD CR=CR              SEND ACK=ON
  RESTORE TABS=ON         ANSWERBACK MODE=OFF     ANSWERBACK CONCEAL=OFF
  WIDTH CHANGE CLEAR=OFF  MONITOR=OFF             TEST=OFF
  ----------------------------------------------------------------------------
                          F3 KEYBRD:
  KEYCLICK=OFF            KEYLOCK=CAPS            KEY REPEAT=ON
  RETURN=CR               ENTER=CR                FUNCT KEY=HOLD
  XMT LIMIT=NONE          FKEY XMT LIMIT=NONE     BREAK=170MS
  LANGUAGE=US             MARGIN BELL=OFF         PRINTER RCV=OFF
  ----------------------------------------------------------------------------
                          F4 COMM:
  DATA/PRINTER=AUX/MODEM    MDM RCV BAUD RATE=9600  MDM XMT BAUD RATE=9600
  MDM DATA/STOP BITS=8/1    MDM RCV HNDSHAKE=NONE   MDM XMT HNDSHAKE=NONE
  MDM PARITY=NONE           AUX BAUD RATE=9600      AUX DATA/STOP BITS=8/1
  AUX RCV HNDSHAKE=NONE     AUX XMT HNDSHAKE=NONE   AUX PARITY=NONE
  (There is a main port (Modem=MDM) and an Auxiliary Port (AUX)
  ----------------------------------------------------------------------------
                          F5 MISC 1:
  WARNING BELL=ON         FKEY LOCK=OFF           FEATURE LOCK=ON
```

```
KEYPAD=NUMERIC          DEL=DEL/CAN             XFER TERM=EOS
CURSOR KEYS=NORMAL      MARGIN CTRL=0           DEL FOR LOW Y=ON
GIN TERM=CR             CHAR MODE=MULTINATIONAL
----------------------------------------------------------------------------
                        F6 MISC 2:
LOCAL=OFF               SEND=ALL                PRINT=NATIONAL
PORT=EIA DATA           SEND AREA=SCREEN        PRINT AREA=SCREEN
DISCONNECT=60 MSEC      SEND TERM=NONE          PRINT TERM=NONE
PRINT MODE=NORMAL       VT100 ID=VT100          POUND=US
----------------------------------------------------------------------------
F7 TABS: You should see several "T" characters spaced 8 dots apart.
    If you don't, hit backspace.
F8 F/KEYS: Normally you will see no definitions for the Function Keys
    here (unless someone has set them up and saved them).  This means that
    they will normally generate their default settings (not displayed here).
    <ctrl><F5> shows the "user defined definition" of the F5 key, etc.
F9 A/BACK: Normally not defined: ANSWERBACK =
F10 EXIT: Selecting "DEFAULT ALL" will make the factory default settings
    the default.
```

HINTS on use of WY−99GT User's Guide: Note that much that is missing from this Guide may be found in the WY−99GT Programmer's Guide. The VT100 emulation (personality) is known as ANSI and uses ANSI key codes per p. A−10+ even though the keyboard may be ASCII. A sub−heading on p. A−13 "ASCII Keyboard" also pertains to VT100 because it has an "ANSI KEY ..." super−heading a few pages previously. But not all ASCII keyboard headings pertain to VT100 since they may fall under a non−ANSI personality super−heading which may found be a few pages previously. Appendix H is the "ANSI Command Guide" except for the VT52 (ANSI) personality which is found in Appendix G.

Wyse 150

When exiting set−up using F12, hitting space changes "no" to "yes" to save the set−up. The sentence to the left of this no/yes is about "vertical alignment" and has nothing to do with this no/yes for saving the set−up (confusing menu design).

END OF Text−Terminal−HOWTO