Remote Serial Console HOWTO

Mark F. Komarinski

mkomarinski@valinux.com

Revision History

Revision 0.1	2001−03−20	Revised by: mfk
First revision		

Most UNIX−based systems have the concept of a serial console. Linux is no exception to this, and this document covers how to set up your hardware to use a serial console.

Table of Contents

1. Introduction

1.1. Copyright Information

This document is copyrighted (c) 2001 Mark F. Komarinski and is distributed under the terms of the Linux Documentation Project (LDP) license, stated below.

Unless otherwise stated, Linux HOWTO documents are copyrighted by their respective authors. Linux HOWTO documents may be reproduced and distributed in whole or in part, in any medium physical or electronic, as long as this copyright notice is retained on all copies. Commercial redistribution is allowed and encouraged; however, the author would like to be notified of any such distributions.

All translations, derivative works, or aggregate works incorporating any Linux HOWTO documents must be covered under this copyright notice. That is, you may not produce a derivative work from a HOWTO and impose additional restrictions on its distribution. Exceptions to these rules may be granted under certain conditions; please contact the Linux HOWTO coordinator at the address given below.

In short, we wish to promote dissemination of this information through as many channels as possible. However, we do wish to retain copyright on the HOWTO documents, and would like to be notified of any plans to redistribute the HOWTOs.

If you have any questions, please contact `<linux-howto@metalab.unc.edu>`

1.2. Disclaimer

No liability for the contents of this documents can be accepted. Use the concepts, examples and other content at your own risk. As this is a new edition of this document, there may be errors and inaccuracies, that may of course be damaging to your system. Proceed with caution, and although this is highly unlikely, the author(s) do not take any responsibility for that.

All copyrights are held by their by their respective owners, unless specifically noted otherwise. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark.

Naming of particular products or brands should not be seen as endorsements.

You are strongly recommended to take a backup of your system before major installation and backups at regular intervals.

1.3. Credits

This HOWTO is based on `/usr/src/linux/Documentation/serial-consold.txt`, written by Miquel van Smoorenburg (`<miquels AT cistron.nl>`). Many thanks to Miquel for the information in his document.

1.4. Feedback

Feedback is most certainly welcome for this document. Without your submissions and input, this document wouldn't exist. Please send your additions, comments and criticisms to the following email address : `<mkomarinski AT valinux.com>`.

2. Why use Serial Consoles?

Serial consoles do not appear to have much going for them. They are slow, require special null−modem cables, and do not provide a graphical interface. But what is going for them is considerable. Serial cables are standard equipment and can run over RJ−45 cables, can run up to 200 ft (about 100m) at 9600bps. Serial concentrators can run the consoles of over 32 ports into a central box, so all the consoles in a cluster can be accessed from a single location.

You will not require a crash cart, KVM switch, or keyboard, montior, or mouse. Because of the serial concentrator, you can access the console of a machine in a colocation cage from your desktop.

The real limiting factor so far has been that even though you can access the Linux console via a serial port, most x86 hardware was not set to send its POST and BIOS information to the serial port. More and more "server" motherboards are starting to include full serial support in the BIOS, so you can access the BIOS and make boot changes via the serial port.

3. Configuring Linux for Serial Consoles

There is two parts to getting a serial console set up under Linux. First, you must tell Linux to redirect its console output to the serial port. Second, you must set up mgetty to start a login process on the serial port once the kernel has completed booting. Some distributions use mingetty for the video console, but mingetty has no serial port support. You will instead want to use mgetty. A third (optional) configuration is to set the hardware BIOS to redirect its POST and BIOS information to the serial port. Check your motherboard documentation for more information about this.

3.1. Configuring LILO and the Linux Kernel

If you're using LILO as your bootloader, you can quickly test using serial console from Linux by entering:

```
LILO: Linux console=ttyS0,9600n8
```

Assuming the LILO tag for your Linux kernel is called "Linux". Change this for the name of your kernel. The generic format for the console option is console=device,options. You can give multiple console statements, and kernel messages will go to all listed devices, but the last one listed will be used as `/dev/console`.

device	The device entry to use as the console without `/dev/`. You can use tty0 to get normal behavior, ttyx to put the console on another virtual console, or ttySx to put the console on a serial port.
options	This is mostly used for passing options to the serial port. The format for this is BPN, where B is speed in bps (so use 9600, 19200, 38400, etc.). The P is parity and is one of three letters: n for no parity, e for even parity, and o for odd parity. The N is the number of data bits, and is usually either 7 or 8. The default is 9600n8. Most users will want to use the default, or increase the speed to 19200 bps.

You should see the Linux kernel start through the decompression process then you will see no more on−screen information until the kernel has completed and mgetty starts up a login prompt on the screen. If you are monitoring the serial port, you'll see the Linux bootup information coming over the serial port. However, you probably will not see a login prompt on the serial port (yet). We'll cover that in Section 3.2.

Once you are sure this is working, you can now edit LILO to pass this information to the kernel each time it boots. You can also configure LILO to send its prompt to the serial port. Fire up your favorite editor of choice and load up the /etc/lilo.conf file. You will want to add two lines, one to the general configuration and one to the specific kernels you want to use.

```
serial=0,9600n8
append="console=tty0 console=ttyS0,9600n8"
```

The append statement contains the statement we listed above, and tells Linux to send its output to the serial port. The serial command goes to LILO, and tells it to open port 0 (ttyS0, or COM1). The options for serial are the same as to the console statement.

> **Note:** Make sure the serial port settings for serial and console are the same. If they are different, you will need to change your serial port application between LILO and the kernel, which becomes very inconvenient.

Re–run `/sbin/lilo` and reboot. You should now see everything except the login prompt on the serial port. Information should still be going to the monitor, just in case you have problems with the serial port.

3.2. Configuring getty for use with serial ports

Some distributions may ship with mingetty that does not support serial ports. The first thing you have to do is make sure the version of getty you are using supports serial ports. Both agetty and mgetty do this. So run off now using your favorite packaging system to make sure this is the case. Don't worry, this document will still be here when you get back.

Back so soon? Great! Let's get that serial port configured.

You will want to make sure that all your serial port settings are consistent. No sense in making getty run at 9600bps, while LILO and the kernel are talking 19200.

To get login prompts to appear on the serial port, edit the `/etc/inittab` file and add a line similar to the following:

```
s0:2345:respawn:/sbin/getty ttyS0 DT9600
```

The format for entries in inittab are covered in most basic Linux and UNIX books, but to repeat, each field is separated by a colon (:) and represent:

- s0 – Arbitrary entry for inittab. As long as this entry doesn't appear anywhere else in inittab, you're okay. We named this entry s0 because it's for ttyS0.
- 2345 – run levels where this entry gets called. If we switch to runlevel 1, this getty process will be shut down.
- respawn – re–run the program if it dies. We want this to happen so that a new login prompt will appear when you log out of the console.
- /sbin/getty ttyS0 DT9600 – the command to run. In this case, we're telling getty to connect to /dev/ttyS0 using the settings for DT9600 which exist in `/etc/gettydefs`. This entry represents a dumb terminal running at 9600bps. There are other entries that run at different speeds.

The entries in `/etc/inittab` will be loaded into **init** when root sends a HUP signal.

```
# kill –HUP 1
```

> **Note:** Remember that **init** always has a PID of 1.

Now that **getty** is set up, you will be able to go from powerup to login prompt all over the serial port

4. Serial Port Applications

This section covers applications and some configuration information that you can use to look at your serial console, now that your Linux boxes are talking to the serial port.

4.1. Minicom

Minicom is one of the easier serial port applications to use. It is curses based, so it's a full screen application with a status bar, menus, and an easy−to−use interface. It is installed on most distributions, and initially has to be run as the root user. In some cases, **minicom** will be installed suid root, so anyone will be able to access the configurations. Check the documentation for your particular distribution to see how it's configured.

Security of minicom is set by the `/etc/minicom.users` file. Usernames can that are listed along with a configuration can use the listed configurations. This allows only authorized users to connect to the serial ports.

Minicom creates individual configurations to separate files. Configure the serial port as needed, then save the configuration. Files are kept in `/etc` with a prefix of minirc.

---

5. Cabling serial ports together

Since you will be connecting two DTEs together, you will need to have a null modem run between the two devices. A null modem crosses transmit and receive, and ties a few status lines together so the application can open the port. This null modem can be a dongle that connects to the cable, or can be built into the cable. A dongle will get expensive if you have a large number of cables, so it is usually easier to get cables with the null modem built in.

Most PC hardware these days use DB−9 connectors, giving 9 pins for transmitting data and status, which is fine for us. Pre−built DB−9 cables can be had for a few dollars for a few feet of cable. More flexible is building a DB−9 to RJ−45 connector and building the null modem into that. The RJ−45 connector then accepts regular 10BaseT cables that can be custom−built, or with varying legths. This gives a lot of flexibiliy in arranging cables, since each cable can be the correct length to run between machines. Little extra cable is left lying around.

DB−9 to RJ−45 connectors can be purchased unassembled since there are no real standards for making this conversion. So long as Tx and Rx cross and CTS RTS cross, you have a null modem connection. The cabling I have here comes from my own design, and works just fine. Note that there have to be two different DB−9 to RJ−45 connectors because of the way pins are switched. I labeled them as "1" and "2". They can be placed on either end of the cable.

Table 1. DB9 to RJ−25 connector

Connector 1		Connector 2	
DB−9	**RJ−45**	**DB−9**	**RJ−45**
1	5	1	5
2	6	2	4
3	4	3	6
4	7	4	7
5	3	5	3
6	2	6	2
7	1	7	8
8	8	8	1
9	n/c	9	n/c