Quick Linux Command Overview

Vince Weaver

http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

October 11, 2013

This document is intended as a quick reference for features available at the command line on a mid-sized embedded Linux board (like a Gumstix or Raspberry-pi).

It's a good idea to try out some of the commands just to see what they can do.

1 The Shell

After you enter your name and password at the login prompt you encounter the "shell". This is where you enter all of the commands.

The default shell is bash, the "Bourne Again Shell" (more computer programmer humor). There are various shells available (bash, sh, zsh, csh, tcsh, ksh) and you can select via `chfn`.

2 Filesystem Layout

There are various directories off of the / root filesystem:

- Executables in `/bin`, `/usr/bin`
- System executables under `/sbin`, `/usr/sbin`
- Device nodes under `/dev`
- Config files under `/etc`
- Home directories under `/home`, also `/root`
- Temporary files under `/tmp`. Often wiped at reboot.
- Magic dirs under `/proc`, `/sys`
- Libraries under `/lib`, `/usr/lib`, sometimes `lib64` too
- Boot files under `/boot`
- User and secondary files under `/usr`. Historically only files needed for boot were directly under /, stuff that can be shared over network (or stored on a second drive if your first drive was too small) would be under `/usr`

- Commercial software / pre-packaged software under `/opt`
- Server storage and other data `/srv`, `/run`, `/var` programs store data
- Default places to mount media (memory keys, CD-Roms, etc.) `/media`, `/mnt`
- If the disk checker finds lost files when fixing a disk after unclean shutdown it may put the files in `/lost+found`

3 Interesting Config Files

Configuration files are stored under `/etc`. Pretty much every program has one, and the setup can be complicated. Here's a list of some common useful ones you find on a barebones system.

- /etc/fstab – the filesystems to mount at boot time
- /etc/passwd – list of all users, world readable
- /etc/shadow – passwords stored here for security reasons
- /etc/hostname – name of the machine
- /etc/hosts – list of local machines, usually searched before resorting to DNS lookup over network
- /etc/resolv.conf – where your nameserver address is put
- /etc/sudoers – list of users allowed to use "sudo"
- /etc/network/interfaces – on debian the network settings are stored here

4 Device Files

Under Linux, interaction with devices is often done by opening various device nodes found under `/dev` and then operating on them via various syscalls (such as `read()`, `write()`, and `ioctl()`).

There are two types of device files, block and char. (Block devices are accessed like an array of bytes with random access, char devices are ones you tend to read linearly). You can manually create device nodes with `mknod` but these days the kernel in conjunction with other daemons create them automatically for you.

- `/dev/sd*` – hard disks (originally scsi disk, but now includes most disks)
- `/dev/tty*` – tty (teletype, logins, serial ports)
- `/dev/zero` – convenience device, always returns 0.
- `/dev/full` – always returns full.
- `/dev/random`, `/dev/urandom` – truly random and pseudo-random numbers
- `/dev/null` – throws away any data you copy into it

Network devices are an exception, you cannot access them through `/dev`.

5 Interesting `/proc` Files

These files are not on disk, but "virtual" and created on-the-fly by the operating system when you request them.

- /proc/cpuinfo – info on cpu
- /proc/meminfo – memory info
- Each process (running program) has its own directory that has info about it

6 Processes

- Each program assigned its own number, a process id, often called a "pid"
- Can list processes with `ps -efa`
- Also can get real-time view of what's going on in a system with `top`
- You can use `kill` to kill (or otherwise signal) a process

7 Commonly Used Commands

- `ls` : list files
  `ls -la` : list long output, show all (hidden) files. on Linux any file starting with . is hidden
  `ls -la /etc` : list all in /etc directory
  `ls *.gz` : show all ending in gz. * and ? are wildcards and can be used as regular expressions.
- `cd DIR` : change directories (folders)
  `cd ..` : go to parent directory
  `cd .` : go to current directory
  `cd /` : go to root directory
  `cd` ∼ : go to home directory
- `cat FILE` – dump file to screen (originally used to conCATenate files together but more commonly used to list files)
- `more / less` – list contents of file but lets you scroll through them. less more advanced version of more
- `exit / logout / control-D` – log out of the machine
- `df / du` – show disk space
  `df -h` pretty-prints it
- `man command` – show documentation (manual) for a command. For example `man ls`
- `rm` remove file. CAREFUL! Especially famous `rm -rf`. In general on Linux you cannot undo a remove.

- `cp` copy file. CAREFUL! By default will overwrite the destination without prompting you.
- `mv` move file. CAREFUL! Can overwite!
  `mv -i` will prompt before overwrite
- `tar` create archive file `tar cvf output.tar dir`
  `tar xzvf output.tar.gz` uncompresses a .tar.gz file
- `gzip / gunzip / bzip2 / bunzip2` compress/uncompress a file. gzip and bzip2 are two common formats, many more exist

8 Compiler and Developer Commands

- `make` – build a file based on list of dependencies in Makefile
- `gcc` – C compiler. Simplest something like this: `gcc -O2 -Wall -o hello hello.c`
- `g++` C++ `gfortran` Fortran
- `as`, `ld` – assembler and linker
- `gdb` – debugger – see Section 20
- `objdump` – disassemble a file with `objdump -disassemble-all`
- `strace` – list system calls
- `git` – source code management

9 Other Commands

- `shutdown` – used to shutdown / reboot
- `last` – list last people to log in
- `su` / `sudo` – switch to root, run command as root
- `uptime` – how long machine has been up
- `uname -a` – show info on the running kernel
- `date` – show the date
  as root you can use `date -s` to set the date
- `whoami` – who are you
- `write` / `wall` / `talk` – write to other users
- `finger` – get info on other users
- `w` / `who` – see who is logged in

- `wc` – count words/bytes/lines in a file
- `dmesg` – print system and boot messages
- `ln` – link files together, sort of like a shortcut
  `ln -s goodbye.c hello.c` – symbolic link. also hard links
- `dd` – move disk blocks around, often used for creating disk images
- `mount` / `umount` – mount or unmount filesystems
- `mkfs.ext3` – make new filesystem
- `e2fsck` – filesystem check
- `ifconfig` / `route` – show and setup network config
- `dpkg` / `apt-get update/upgrade/install` – debian only package management
- `ssh` / `scp` – log into other machines, copy files remotely
- `lynx` – text-based web browser
- `reset` – clear the screen and reset settings (useful if you accidentally cat a binary file and end up with a screenful of garbage). Control-L also refreshes the screen
- `linux_logo` – my program
- `adduser` – add a user to the system

10 Editing files

Linux and UNIX have many, many editors available. Most famous are vi and emacs. On our board using nano might be easiest.

- `nano` – a simple text editor.
  `nano FILENAME` – edit a filename
  It shows the commands you can do at the bottom. `^O` means press control-O
  control-O : writes
  control-X : exits
  control-W : searches
  control-\: search and replace
  control-C : prints line number
  control-K : cuts lines
  control-U : pastes recently cut lines

11 Redirection and Pipes

- redirect to a file : `ls > output`
- redirect from a file : `wc < output`
- pipe from one command to another : `ls | wc`, `dmesg | less`
- re-direct stderr : `strace 2> output`

12 Job Control

And by job we mean the same thing as a running program or process.

- Press control-C to kill a job
- Press control-Z to suspend a job
- Type `bg` to continue it in the background
- Type `fg` to resume it (bring to foreground)
- Run with & to put in background to start with. (ie, `mpg123 music.mp3 &`).

13 Permissions

Files have various permissions that say who can access them.

Usually they are owned by a user or group. Additionally they have read/write/execute permissions.

You can see the permissions on a file with `ls -l`. To the far left are the permissions; the first column are the user (your) permissions, the middle group (everyone in your group) permissions, and the last is global (everyone) permissions.

- user, group – use `chgrp` to change
- read/write/execute – use `chmod` to change

14 Shell Scripts

It is possible to quickly write a script that executes a series of shell commands. Just put the commands in a file and mark it executable. It's possibly to make very complicated scripts, as the shell contains a full scripting language.

- Create a list of files in a dir
- Start with the shell, `#/bin/sh` (or perl, etc)
- Make executable `chmod +x myfile`

15 Command Line History

It is useful when doing a lot of command line work to access previously typed commands or have the commands automatically completed.

- Can press "tab" to auto-complete a command
- Can press "up arrow" to re-use previous commands
- Can use "control-R" to search for previous commands

16 Environment Variables

These variables go with your shell, and programs you run can access them.

- `env` lists all the current variables.
- How to set these varies from shell to shell. With bash `export TERM=vt102`
- The `PATH` variable shows the search path for executables. You'll note "." isn't in the path. This is for security reasons. If you were in another user's directory they could do things like create a hostile program called `ls` and then if you typed `ls` it would run their code rather than the real command. This is why when you compile a program, to run it you have to prepend it with `./`.

17 Configuring Sudo

When doing embedded work it is handy to be able to run commands as root without having to keep using "su" to become root. In this case it is useful to set up the "sudo" command. With that you can run a single command as root, entering only your user password. Something like `sudo ls /root`.

To do this, either run `visudo` or open the `/etc/sudoers` file. Add an entry toward the bottom with `vince ALL=(ALL:ALL) ALL` replacing "vince" with your username.

18 Package Management

Linux programs are often distributed as "packages" which are file archives holding all of the parts needed to install a program. This includes the binaries, config files, libraries, documentation, etc. They also typically containing a list of dependencies on other packages that will need to be installed.

Two common file types you'll find are `.rpm` and `.deb`. rpm is used by RedHat, Fedora, SuSE and a few others. deb is used by Debian and Ubuntu. You may also find .tar.gz (or .tar.bz2 or .tar.xz) files. These usually contain source code (which you typically install by `tar -xvf file.tar.gz ; cd file ; ./configure ; make ; sudo make install`) but they also sometimes carry binary files that can be installed by an included script. Read documentation that comes with them for how to install.

I will talk about Debian systems here, as that's what I typically use. There are tools called `dpkg` and `apt` which handle package management for you. If you get a `.deb` file you can run `dpkg -i file.deb` to install it. Better is to use `apt`.

On your system run `sudo apt-get update` to update the local list of all packages available. You can run `sudo apt-get upgrade` to upgrade any packages that need it. To find packages that are available you can run `apt-cache search keyword`. To install a package from that list do something like `sudo apt-get install packagename` and assuming you have a network connection it will download and install the program and any dependencies for you.

19 Using Minicom to Connect to a Serial Port

When using an embedded system, one does not always have the luxury of a keyboard/display or even a network connection. If you're lucky you might have a serial port you can connect to.

You can use minicom to connect to these devices. The commands for minicom can be a bit obscure, it is based on the old DOS Telix program.

First find out which serial port you are using. On older machines with an actual serial port it is something like /dev/ttyS0, but these days most people are using USB/Serial devices. Plug in the USB cable and use `dmesg` to check the logs, it will usually be something like `/dev/ttyUSB0`.

Start minicom, you can use something like minicom -D /dev/ttyUSB0.

Configure the settings. Control-A followed by Z will bring up a help menu. You can set the serial port to the proper speed, usually 115200 8N1.

If backspace in the terminal doesn't work, and full-screen apps are corrupted, you might have to set the terminal emulation. Try VT102, so be sure minicom is in that mode, and at the prompt type `export TERM=vt102`.

If you get output from your board, but typing doesn't work, try going into the serial port settings and flipping the values for HW flow control and SW flow control.

To transfer files, the easiest way is to install the `lrzsz` package.

To send a file to the board, in the window type `rz` and in minicom type Control-A S. Select file to send by navigating the menu and pressing space bar twice to select.

To receive a file from the board, in the window type `sz filename` and in minicom Control-A R.

20 Debugging with gdb

The debugger most commonly used on Linux is `gdb`. Unlike the nice integrated-debugging environments you can get for Windows, `gdb` is a hard-to-use command-line program. (There are graphical frontends you can get, but they don't necessarily work well in an embedded environment).

For this example, let's say we have a program called `test1` that we want to debug.

To debug it, run `gdb ./test1`. This should give you a gdb prompt.

To run within the debugger, type `run x y z` and replace x y z with any command line arguments you might want (or leave blank).

If your program crashes, gdb will stop and give you a prompt and you can debug from there. Otherwise you will need to set a breakpoint. This can be tricky. To add a breakpoint at a function call you can do something like `break read`. Replace read with the name of the function you want. If you're programming in assembly you can also set breakpoints on label names. If you want to set a breakpoint on an address, it is something like `break *0xdeadbeef` replacing 0xdeadbeef with the address you want.

Once your program is stopped, you can do many things. A sampling:

- Dump the registers `info regis`

- Disassemble the current code block `disassem`
- Step forward one function call `step`
- Step forward one assembly language instruction `stepi`
- Examine a memory address `x 0xdeadbeef` replace 0xdeadbeef with the address of interest.

To continue a program to the next breakpoint, type `cont`.
To quit, just type `quit`

When a program crashes, you can have it generate a "core dump" containing the state of the program at crash, and load it into the debugger. You first have to enable the coredump, something like `ulimit -c unlimited`. Then when the program crashes, it will say something like `segmentation fault, core dumped` and a `core` file is created. Then run `gdb ./prog_name ./core` (replacing with your proper program name for prog_name) and gdb will start with all the info from the time of the dump.