adafruit learning system

An Illustrated Shell Command Primer

Created by Brennen Bearnes

Last updated on 2018-08-22 03:45:56 PM UTC

Guide Contents

Introduction

This guide assumes you have access to the shell on a Linux computer. All the examples use a Raspberry Pi running Raspbian. If you haven't, you should start with What is this "Linux", anyhow? (https://adafru.it/jDZ) and What is the Command Line? (https://adafru.it/sdo)

Choosing Your Own Adventure

This is Adventure (https://adafru.it/ek5), a game almost as old as Unix itself, and much older than Linux:

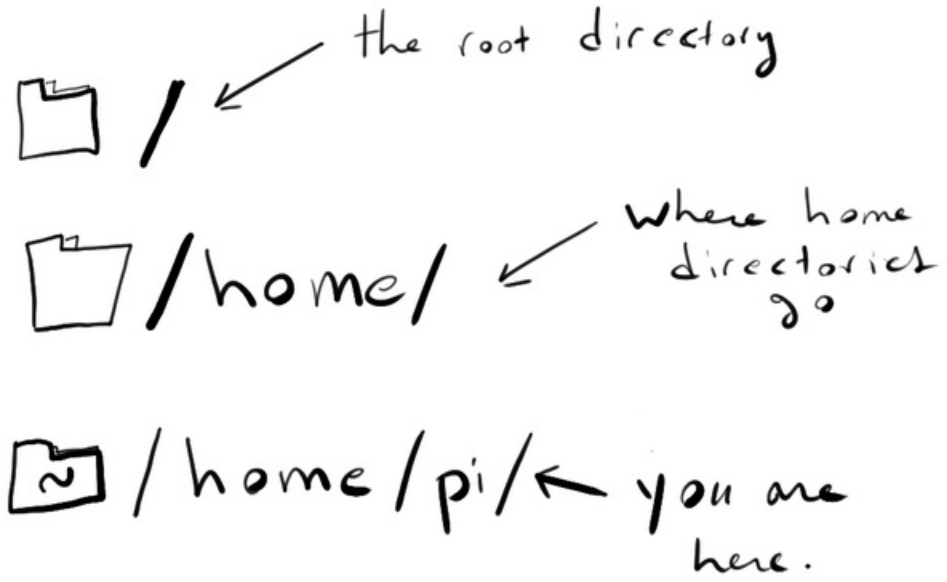
Like the shell, *Adventure*'s interface is made entirely of text.

You interact with the world of the game by typing commands and reading responses. ENTER BUILDING or GET LAMP (https://adafru.it/ek6), for example.

Exploring the Machine

The shell is a lot like that game world, except that instead of a map full of rooms and objects, you navigate a computer's **filesystem** and work with the files it contains.

The filesystem, everything the operating system needs to boot up, run software, save files, take photos, etc. is organized into **directories**. We usually call these "folders" in the Windows and Macintosh worlds, but the concept is the same. A directory is just a special kind of file that can contain other files, including directories.



As a user, you have a **home directory**. In the shell, ~ (the tilde (https://adafru.it/ek7)) is a common shorthand for this. In this case, it points at /home/pi.

Log in to your Pi, acquire a terminal, and verify this:

```
pi@raspberrypi ~ $ pwd
/home/pi
pi@raspberrypi ~ $
```

`pwd` stands for **p**rint **w**orking **d**irectory. Commands in the Unix tradition tend to have short, cryptic names. Some can be easily remembered because they stand for something obvious. Others just have to be committed to memory.

If you have a look around your Pi, you'll find that it contains all sorts of stuff. You just need to know a few more commands. The following pages provide quick introductions to some of the basics.

Listing Files: ls

`ls` is used to **l**i**s**t files (note the first letter is the letter L not the number 1!)

Let's try it from our home directory:

Notice that second command and its output?

`ls -a` shows **a**ll the files in a directory, including hidden ones. Why are there hidden files? Because in the Unix world, all directories contain two special sub-directories:

`.` points to the current directory - `/home/pi` is the same as `/home/pi/.`

`..` points to the parent directory, the one that holds the current directory - `/home/pi/..` is really just `/home/`

`ls` normally hides everything starting with a dot, because these don't really convey any extra information to a user.

To complicate matters even further, it's traditional for special configuration files to be stored in `~` and named like `.something` so that they won't show up in a directory listing by default.

Want to see more detail about individual files?

The `-l` option, for a **l**ong listing, will give you several columns of useful data. For the file called `.bashrc`, you have:

```
-rw-r--r--  1 pi   pi    3243 Sep  8 20:23 .bashrc
```

`-rw-r--r--` is a shorthand for the file's type and **mode** or **permissions**. In this case, the file is a plain old file (not a directory or other special file) and:

- **r**eadable and **w**riteable by its owner
- **r**eadable by its group
- **r**eadable by everyone

The rest of the line, in order:

- `1` is the number of hard links to the file
- `pi`, repeated twice, tells you the **owner** and **group** of the file
- `3243` is the number of bytes taken up by the file
- `Sep 8 20:23` is the file's modification time

You don't need to worry too much about this stuff yet - just remember that `-l` gives you more detail.

Changing Directories: cd

In order to move around between directories, you use `cd` for **c**hange **d**irectory.

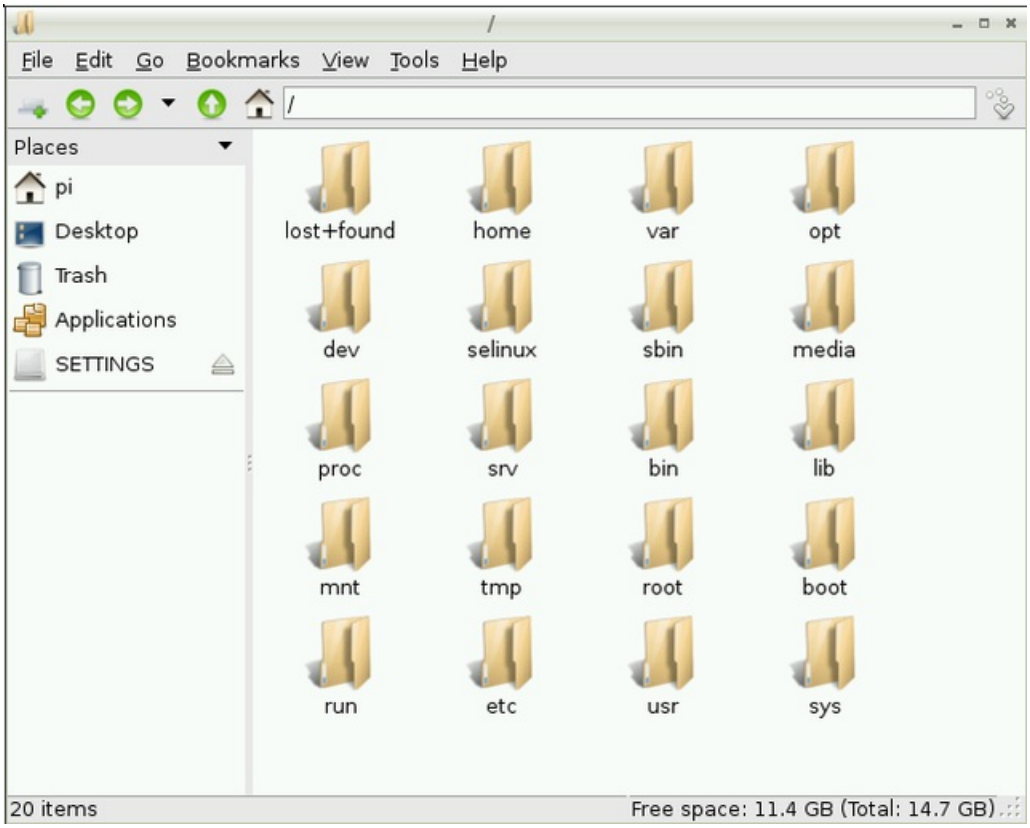
You can give cd **relative paths**, like so:

Remember that ".." is a shortcut for "the directory above this one".

You can also use an **absolute path**, like so:

Remember that `/` is the **root** of the filesystem, the top-level directory under which everything else is nested.

The default graphical file manager in Raspbian displays the root directory in a way that's probably familiar from other operating systems:



Double-clicking on a folder icon here is equivalent to typing `cd foldername`. Putting a full path in the location bar at the top, like "/home/pi", is equivalent to typing `cd /home/pi`.

Spend some time looking around the root directory. (There's a lot there, and like the names of commands, it can be pretty cryptic at first. Don't worry if it seems a bit overwhelming; you're just getting the lay of the land.)

Some interesting places:

- `/etc` is full of system-wide configuration files
- `/proc` and `/sys` are full of information about running programs and the kernel
- `/var` contains things like logfiles that the system writes during the course of operation
- `/dev` contains files that map to devices (like drives, network interfaces, and virtual terminals) attached to the system

Looking Inside Files: cat, less, head, and tail

I said "look around the root directory", but I left out some important tools for doing that. You already know about listing the contents of directories with `ls`, but what if you want to look at the contents of an individual file?

In a graphical file manager, you might double-click an icon and let your desktop environment decide the appropriate application for opening that kind of file. The shell takes a much different approach. What command you use on a file depends entirely on what you want to accomplish.

cat

`cat` has nothing to do with cats. It's short for **cat**enate, and it dumps the contents of text files. Cats think about lots of things, but not usually about text files.

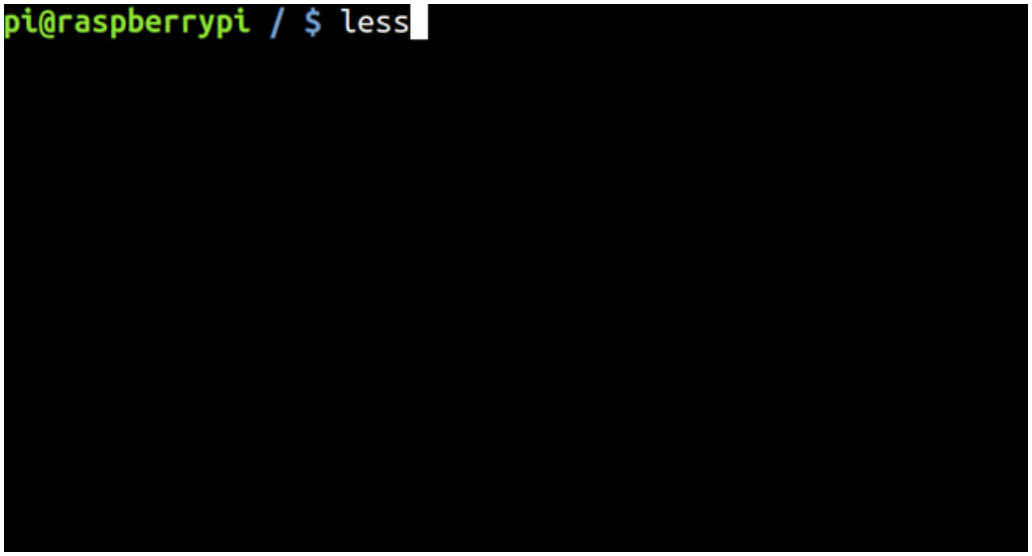
For example, here's one that contains basic information about the system's processor:

less

Sometimes you'll want to look at a really long file, one that takes up more room than you've got in your terminal.

`less` is what's known as a **pager**. It will display one page of a file at a time and let you scroll up and down through the file at your leisure.

`/usr/share/dict/american-english` is a long list of known words in American English. (If that's not available, try `/usr/share/dict/words`, which should point to the dictionary for your Pi's locale.) Try opening it in `less`.

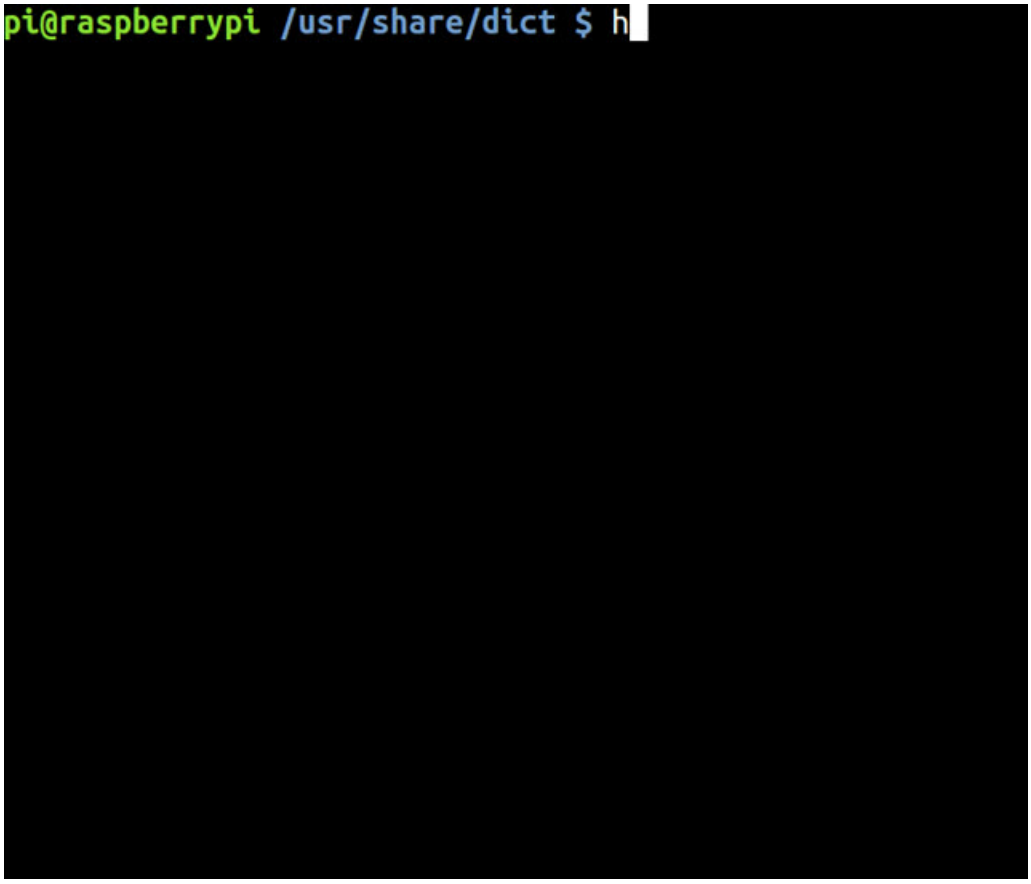


Within `less`, a *ton* of different keys are available. The basics are easy to remember, but the online help lists all sorts of advanced features.

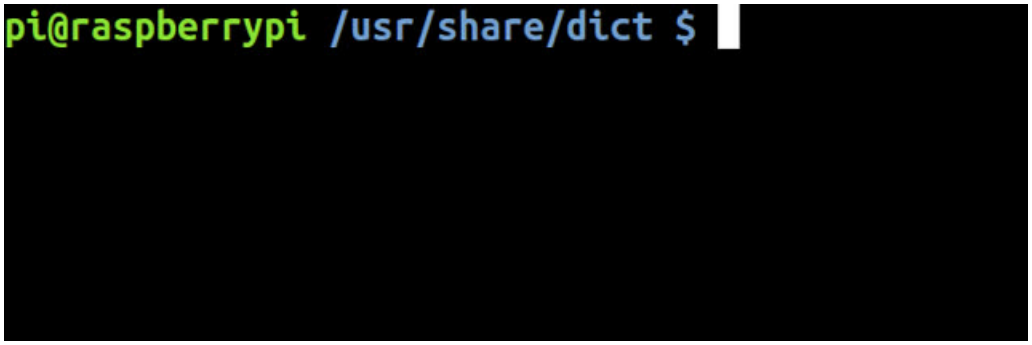
space	advances by one page
down and up arrows j and k	scroll by one line
/	searches for text
q	quits to the shell
h	displays help screen

head and tail

Sometimes you just want a quick look at the beginning or end of a file. This is useful for getting a sense of the contents of very large files, and for seeing the latest additions to things like logs that routinely have new data appended to them.



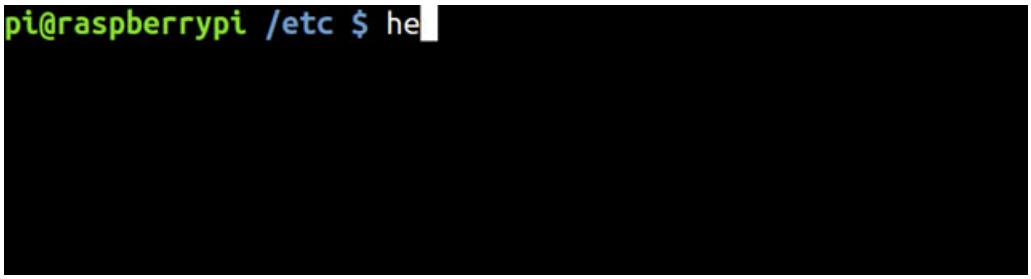
If you want to see a different number of lines than the default, you can specify:


pi@raspberrypi /usr/share/dict $

Attaining Superpowers: sudo

By now, it's possible you've stumbled across a file that seems like it's off-limits to you. That's a little weird, right? It's your computer - why can't you see everything?

It turns out that you can. You just need to assert some authority, which is where `sudo` comes in.



You'll often need to use `sudo` for changing system-wide configuration files, looking at things that have security implications, or installing new software.

We'll address each of those topics in more detail, but for now just remember that the formula is:

```
sudo [your command here]
```

If you're using `sudo` for the first time, or for the first time in a few minutes, you'll be prompted for your password.

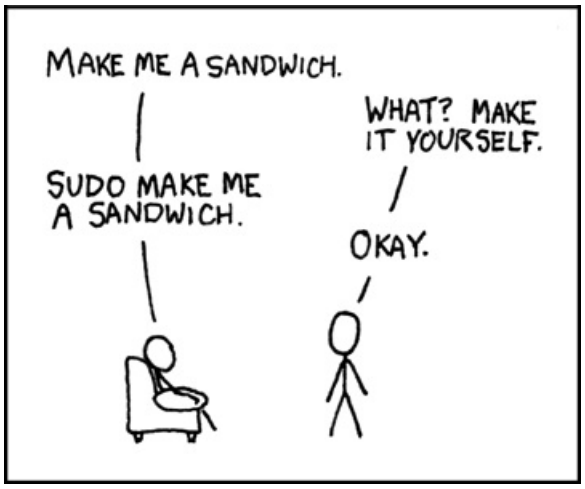
sudo can get you into trouble

If you're thinking that the access restrictions on some files and actions must exist for a reason, you're right. Raspbian is configured by default to be a pretty wide-open system, with *doing stuff* taking higher priority than *being really secure about stuff.* Needing to use sudo should always be taken as a sign that you *could* break something.

On any system, always try to make sure you understand what a command is doing before you run it, or at least be pretty sure you trust the person or website telling you to run it.

obxcd

One great side effect of this tutorial is that now you will 'get' Unix nerd joke comics like this one from xkcd.com (https://adafru.it/el2) (a Mecca of unix nerd joke comments):

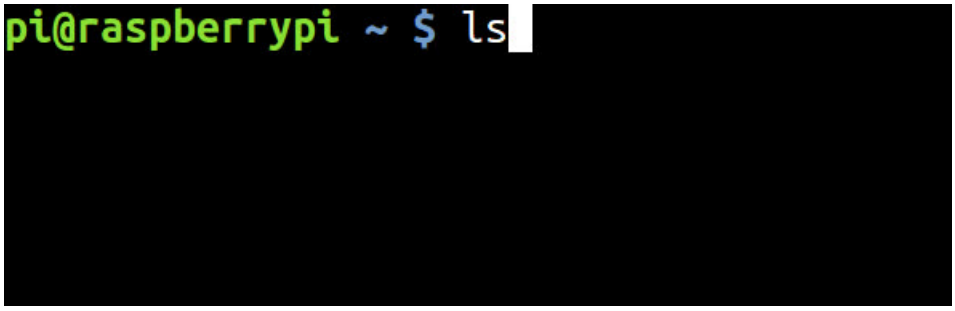

MAKE ME A SANDWICH.
WHAT? MAKE IT YOURSELF.
SUDO MAKE ME A SANDWICH.
OKAY.

Creating Directories and Files: mkdir and touch

mkdir

Suppose you want to create a directory of your own?

I often keep one called "notes" in my home directory, which in turn contains some text files.

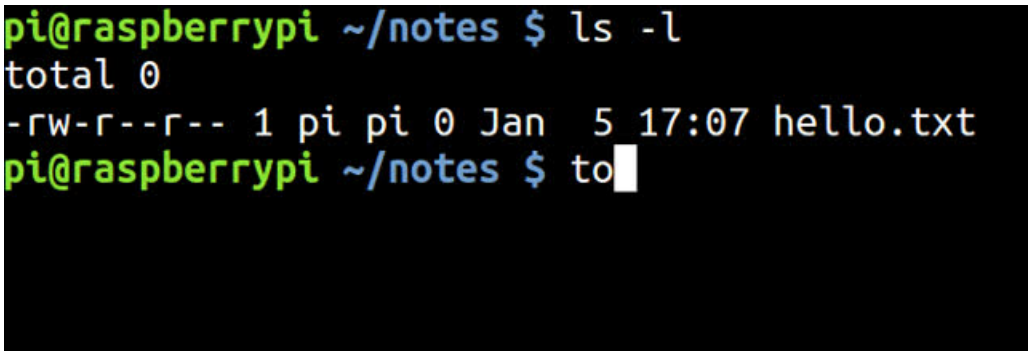


touch

Suppose we want to add a `hello.txt` to that directory?

`touch` doesn't put anything *inside* that file. It just creates it.

There's one other thing to know about `touch` : If you run it again a bit later on the same file...

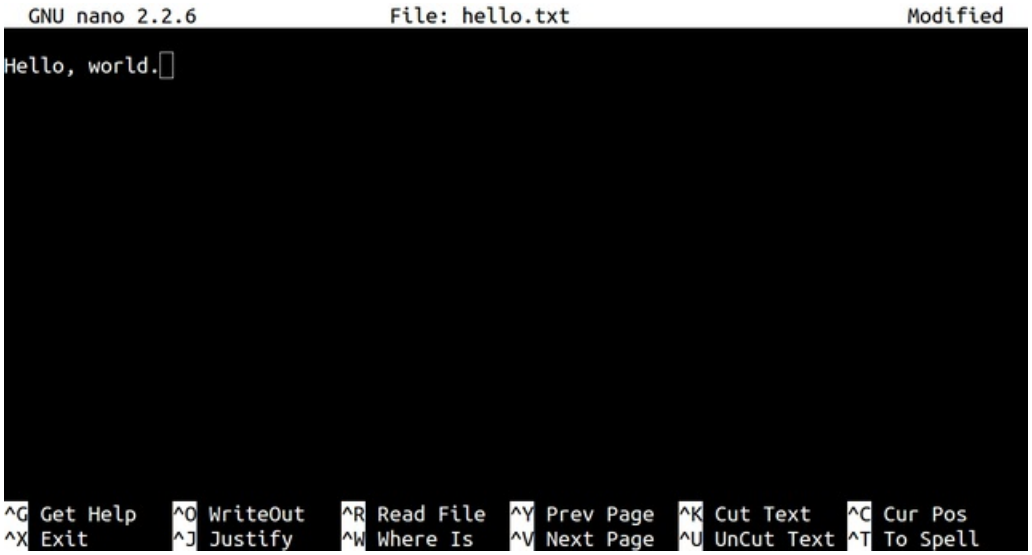


...you'll update the timestamp. This might seem like a trivial thing, but it comes up surprisingly often.

Of course, this is obviously sort of a contrived example. If you're going to have text files, you probably want to put something *in* them. This is where an editor comes in.

Editing Files: nano

`nano` is a little text editor. You can think of it as something like a terminal version of Windows' Notepad.



The great thing about this interface is that it tells you exactly what to do. The bottom two lines of the screen are a cheatsheet for the most commonly used commands.

Just remember that `^` stands for "press the Ctrl key at the same time as". Ctrl-X will quit, Ctrl-O will save a file, etc. It doesn't take long to memorize these, but it's nice not to have to guess.

Your Friend the Text Editor

Text editors may not seem like the most interesting software, but you'll quickly find that an editor is indispensable for many of the things you want to accomplish in a shell environment.

`nano` is really sort of a baseline editor: It gets the job done, and it's easy on new users. Most modern systems will have it installed by default.

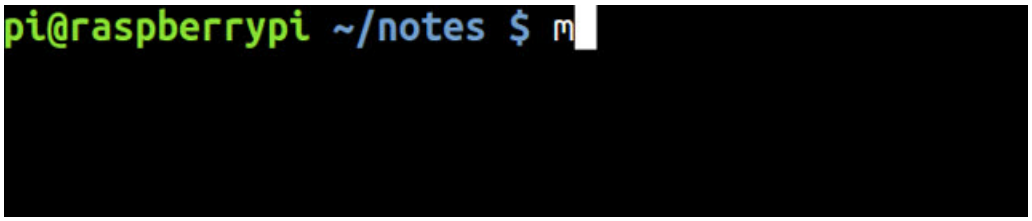
There are substantially more powerful choices that work well in a terminal, like Emacs (https://adafru.it/ek8) and Vim (https://adafru.it/ek9), but these come with a steeper learning curve, so for the moment we'll use `nano` in our examples.

Moving, Renaming, and Copying Files: mv and cp

mv

There are two cases in which you're going to need `mv`: One is that a file is in the wrong *directory*. The other is that the file itself has the wrong *name*.

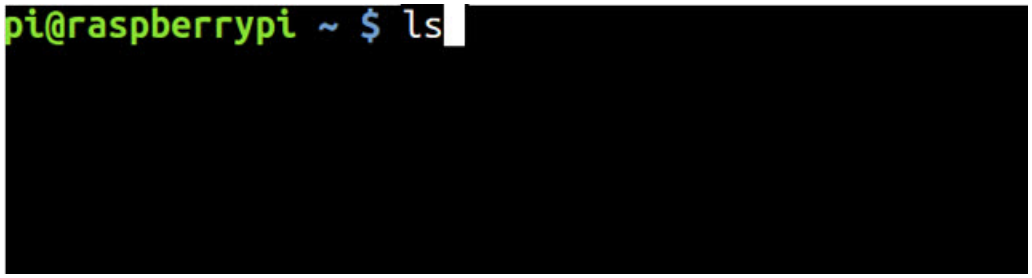
Suppose you've decided that `hello.txt` doesn't really belong in your notes directory.



In this example, we say "move hello.txt up one directory level".

Instead of using `..`, we could have specified `/home/pi`. It's just faster to type a couple of dots. It would have been even faster to type `~`.

Now suppose you've decided that `hello.txt` itself is called the wrong thing, and should be `hello_world.txt` instead.



Remember that a directory is just a special kind of file. `mv` will work to move entire directories just the same as individual files.

> moving files can overwrite other files

Always make sure you're not going to overwrite something important when you move a file. There are a couple of good flags to know in order to avoid this:

mv -i	For interactive - mv will ask what you want to do before overwriting any files.
mv -n	For "no clobber". Will skip overwriting any files.
mv -b	Make a backup of any files that get replaced.

cp

In order to duplicate a file, use `cp`:

```
pi@raspberrypi ~ $ ls
Desktop  hello_world.txt  notes  python_games
pi@raspberrypi ~ $ cp hello_world.txt other_file.txt
pi@raspberrypi ~ $ ls
Desktop  hello_world.txt  notes  other_file.txt  python_games
pi@raspberrypi ~ $
```

Trying to copy a directory, however, behaves oddly:

```
pi@raspberrypi ~ $ cp notes old_notes
cp: omitting directory `notes'
```

By default, `cp` skips over directories. Why? Well, there's some good discussion about this (https://adafru.it/eka) over on Stack Exchange, but the short answer is probably that copying everything in a directory could be a very expensive operation. (Imagine if a directory contained many thousands of files.)

In order to copy a directory, we need to give `cp` the `-r` flag to enable **recursion**. This is part of a broader pattern: In order for many commands to work on entire directories full of files, recursive operations must be specified with a flag, usually `-r` or `-R`.

copying files can overwrite other files

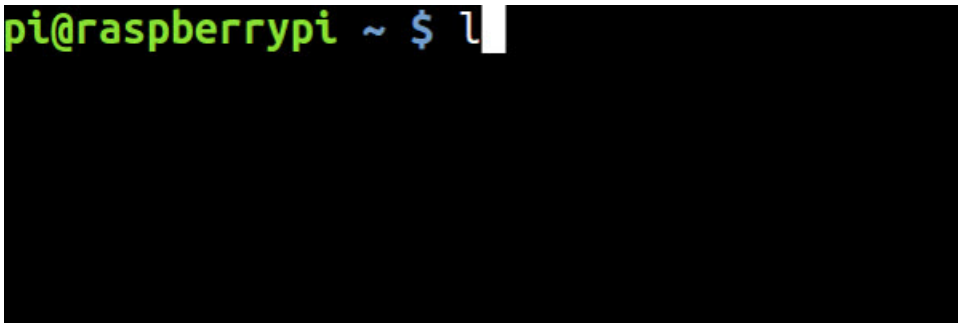
As with `mv`, always make sure you're not going to clobber anything important by copying over it. The `-i`, `-n`, and `-b` flags work here too.

Deleting Files and Directories: rm

> WARNING: Here be dragons!

If you want to outright **re**m**o**ve a file, you'll need `rm`. For individual files, just specify the filename(s):

Just like cp, if you want to get a directory, you'll need `-r` for **r**ecursive. `-i` for **i**nteractive also works, and I often use it if I want to be really careful about what I'm doing.



> you'll shoot your eye out!

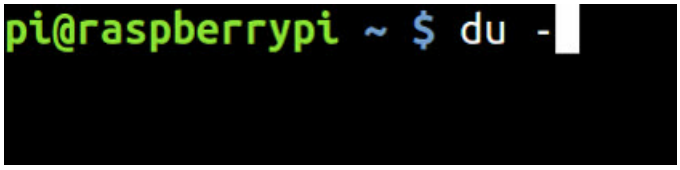
You should always be careful with `rm`, and doubly so with *recursive* rm. It's always a good idea to make sure you know what directory you're in, where the thing you want to delete lives, and that you're *sure* you want to delete it.

Unlike graphical file managers, `rm` doesn't come with an undo button or a Trash where you can retrieve things. Once a file is gone, you will only be able to get it back at considerable effort, if at all. **Tread lightly!**

Disk Space, Memory Use, and CPU Load: du, df, free, and w

du

It's often necessary to figure out how much space your files are using up, especially on smaller devices like the Raspberry Pi, where storage is frequently limited. This is where `du` (think **d**isk **u**sage, even though your "disk" is probably an SD card) comes in. By default, the output is pretty verbose and hard to read, so I usually use `-h` for **h**uman readable numbers with units and `-s` for a **s**ummary.



You can also specify a path, and without the `-s` it will tell you the size of every file it looks at.

df

Sometimes it's easier to come at the question by checking how much space is *left* on a drive.

`df` (**d**isk **f**ree) provides a quick summary, broken out by device and where that device is attached to:

free

A resource even more constrained than storage on the Raspberry Pi is RAM. `free` provides a useful quick summary of the state of the computer's memory:

Again, `-h` gets you **h**uman-readable numbers with units. Here's a good breakdown of how to read those numbers (https://adafru.it/ekb).

w

It can also be useful to know **w**ho's logged in, the system's uptime, and CPU load average for the last 1, 5, and 15 minutes. That's the grab bag of info supplied by `w`:

You can also get just that first bit by running `uptime`.

What is load average?

That's kind of a tricky question. The first thing you need to know is that higher numbers mean more load. The second thing is that it matters how many processors you have - mentally divide the number you see by the number of processors, and that's the number you should worry about.

What's a number you should worry about? Well, it depends. Here's a good detailed explanation.

Further Reading

The shell's part of a whole culture with its own history and literature. There's no one right way to learn about something like that, any more than there's one right way to approach human culture in general. Here're some suggestions for other things that might be helpful.

Perhaps unsurprisingly, Wikipedia has a lot of good information about command-line utilities. You could profitably start just about anywhere near their Unix (https://adafru.it/ekd) entry. For example, this list of Unix commands (https://adafru.it/eke).

There's a Unix & Linux Stack Exchange (https://adafru.it/ekf).

Earlier (https://adafru.it/CeL) I mentioned *The Unix Programming Environment (https://adafru.it/ekh)*, by Brian Kernighan and Rob Pike. It's that rare technical book that's worth your time more than 30 years after its first publication.

My edition of *Linux in a Nutshell (https://adafru.it/eki)*, by Ellen Siever et al., is a decade or more out of date, but I still flip through it from time to time and learn something new.

And last, I've been working for a while now on *userland: a book about the command line for humans (https://adafru.it/ehQ)*. If this guide just isn't doing the trick, *userland*'s more literary and detailed take on the subject might be worth checking out. (On the downside, it doesn't have nearly as many animated GIFs.)

That's just about it for now, but stay tuned - from here we'll be tackling more advanced shell usage:

- An Illustrated Guide to Shell Magic: Standard I/O & Redirection (https://adafru.it/CeM), which covers features like pipes, redirection, and standard IO that allow us to stitch little commands together - the true power of the shell.
- Aliases, wildcards, loops, and other techniques for typing less and accomplishing more.
- Writing scripts to solve larger problems.
- What it means when we say that "everything is a file".
- System administration tasks like upgrading and installing software via `apt` and other package managers.
- Data munging for fun and profit.