Pantry User Guide

Omari Norman

Pantry User Guide

Omari Norman

Version 31 released Sun, 25 Jan 2009 14:37:38 -0500.

Data in the `starter_database` file is derived from U.S. Department of Agriculture, Agricultural Research Service. 2008. USDA National Nutrient Database for Standard Reference, Release 21. Nutrient Data Laboratory Home Page: http://www.ars.usda.gov/ba/bhnrc/ndl This data is in the public domain and there is no copyright, see http://www.ars.usda.gov/Main/docs.htm?docid=6233#copyright

Table of Contents

List of Examples

About Pantry

Pantry is a command-line oriented nutrient analysis program. It can help you find the nutrient content of foods and recipes. You can also use it to track and analyze your nutrient intake.

Pantry is a true command-line oriented program. There are no menus or prompts. Of course this is clearly different from Web-based or GUI programs, but it also differs from programs that run in a text console but are menu-driven (such programs have been called text user interface [http://en.wikipedia.org/wiki/Text_user_interface] programs).

Pantry may or may not be what you are looking for, so here is a rundown of the advantages and disadvantages of this program.

Pantry's advantages

Pantry's true command-line interface gives it many advantages. Because Pantry works from your shell prompt, you can easily combine it with other text-processing tools. You can also easily write scripts incorporating Pantry, in ways that even I cannot anticipate. This is the strength of the Unix "toolbox" way of using a computer.

In addition, nothing beats the speed of a command-line program for something you use frequently and are familiar with. If you are using a nutrient-analysis program to track your daily food intake, you will appreciate how quickly you can use Pantry for this purpose. Indeed, I developed Pantry due to my frustration with current tools because it was very tedious to use them to quickly tally a day's food intake.

Because Pantry runs from a text console, you can easily set it up on one computer that has an SSH server running. You may then access your nutrient data from any computer that has an SSH client. Though Web-based nutrition-tracking services also offer the ability to access your data from any Internet-connected computer, Pantry is more private than Web-based services.

Pantry's disadvantages

The biggest disadvantage of using Pantry is the same as its biggest advantage: its command-line interface. Graphical user interface programs attempt to be self-documenting: just sit down, click on some buttons, and hopefully you can figure things out. With Pantry, on the other hand, you will absolutely have to read this manual to figure out how it works, and you will need some practice before you are comfortable with Pantry. In this way, Pantry resembles other command-line oriented Unix programs. As with other Unix programs, once you learn Pantry, you will love its speed and efficiency--but you will have to spend some time learning.

Similarly, because of its command-line interface, you will find that you are most efficient with Pantry if you know your way around a Unix shell prompt. For example, you will find that you can use Pantry more quickly if you know how to use your shell's features to manipulate your command history. Such knowledge is useful for any Unix command-line program, not just Pantry; however, building up this knowledge takes some time.

Pantry has no tools to graphically visualize your food intake. I might eventually add such features using Gnuplot [http://www.gnuplot.info/] or something similar.

Alternatives

I know I like to check out many possible programs before settling on one to learn to use--and I always like trying new programs too. Here are some possible alternatives to Pantry that you might take a look at:

NUT	This is probably the leading nutrient analysis program for Unix. It has a text user interface and is very mature. NUT website [http://nut.sourceforge.net].
Crosstrainer	The best GUI program for nutrient analysis that I have ever seen. It does more than nutrient analysis, too--for example it can track your exercise too. (Pantry will never do anything other than nutrients.) Crosstrainer only runs on Windows though, and I have not used Windows in quite awhile. Crosstrainer is also somewhat expensive. Crosstrainer website [http://www.crosstrainer.ca/].
nutritiondata.com	This website has great tools, and neat graphical features. Using it as a food diary is somewhat cumbersome, however. nutritiondata.com website [http://www.nutritiondata.com].
Fitday	This is the best food diary website that I have found. Fitday website [http://www.fitday.com].

Pantry on the Internet

Visit the Pantry home page at http://www.smileystation.com/pantry. I post new versions there. I also update the Pantry Freshmeat listing at http://www.freshmeat.net/projects/pantry when I release new versions. Freshmeat has subscription services that will notify you when there is a new release.

Pantry's source code is tracked with Git; you can clone the repository from Github [http://github.com/massysett/pantry/tree/master].

About the documentation

Right now you are reading the Pantry User Guide. This is a narrative document to teach you step by step how to use Pantry. I hope you'll read it from beginning to end. In the PDF version, some of the examples run off the right side of the page. If you're curious about what you're missing (typically very little of substance) just look at the HTML version.

Also, like any good[1] Unix program, Pantry has manual pages. They are available by invoking "man pantry" or "man pantryd" at the command prompt. The manual pages are not intended to teach you how to use Pantry. Instead, they serve as reference guides if you wish to jog your memory about how a particular feature works. You will also find the manual pages at the end of this User Guide.

[1] I don't mean to presume that Pantry is a good program, but I have tried to imitate the best where I can.

Installation

Pantry is written in C++ [http://en.wikipedia.org/wiki/C_plus_plus] and distributed using the GNU Autotools [http://en.wikipedia.org/wiki/GNU_build_system]. If you know what that previous sentence means, then you don't need to read the rest of this section--install Pantry the usual way.

If on the other hand you have never compiled software, never fear. Sometimes compiling software from source code can be tricky, but Pantry is quite easy to compile and install. This section will lead you through the process step by step.

Pantry should compile and run perfectly on any recent Unix-like operating system. This includes any Linux system, the BSDs, Mac OS X, and any of the proprietary Unices. It might also compile and run in Cygwin; I am unfamiliar with Cygwin so I can't offer any help there. Pantry will not compile or run in Microsoft Windows, and I doubt this will ever change.

Install everything you will need to compile Pantry

You'll need to have the following software installed before you attempt to compile Pantry. All this software is quite common and will be readily available on any Unix system. On a Linux system, use your distribution's package management system to install this software, if it is not installed already.

make — **make** invokes the compiler, and it also installs the compiled binaries (and other files) in their proper locations. On Linux this package is typically named make.

The standard C library — The standard C library contains a variety of essential routines that Pantry uses for file operations and interprocess communications. I can guarantee that the standard C library is already installed on your system, as no Unix system can work without one. However, on many Linux distributions you will need to install the additional "development" files. These files are necessary if you wish to compile software from source. Look for a package named libc6-dev or glibc-devel or something similar. Most other Unix systems will already have these additional files installed.

A C++ compiler — Pantry is written in C++ so you will need a C++ compiler. Pantry uses only standard features of the C++ language, so any recent C++ compiler will work. On Linux, install the GNU C++ compiler. Look for a package named g++, gcc-g++, or something similar.

C++ standard library — All C++ programs require the standard C++ library. Though the standard C++ library likely is already installed on your system, you may need to install the additional "development" files, especially if you are on Linux. Look for a package named libstdc++-devel, libstdc ++-dev, or something similar.

Compiling and installing

Now that you've installed all the prerequisites, compiling and installing Pantry is simple:

1. In your normal user account, unpack the tarball with **`tar -xzf pantry-31.tar.gz`**

2. Change to the newly created directory with **`cd pantry-31`**.
3. Run **`./configure`**.
4. Run **`make`**.
5. Become the root user by issuing **`su`** or **`sudo -i`**.
6. Install Pantry with **`make install`**.

If you are curious about what the commands listed above are doing, consult one of these tutorials about how to install software from source code:

- Tuxfiles [http://www.tuxfiles.org/linuxhelp/softinstall.html]
- Linux Howtos [http://liquidweather.net/howto/index.php?id=82]

Getting help and reporting bugs

The best place to start when you're learning Pantry is this user guide, which was written to teach you everything you need to know. The manual pages, pantry(1) and pantryd(1) were not intended to teach you how to use Pantry; they're better for jogging your memory on how to use this feature or that.

If you're having trouble installing or using Pantry, send me an email at omari@smileystation.com [mailto:omari@smileystation.com]. Also, please send me an email if you have any suggestions for improvement or have any bugs to report.

Chapter 1. The Pantry Paradigm

Every software package comes with a set of terminology and ideas that you need to learn in order to use the software most effectively. For example, emacs users are familiar with buffers, regions, and the kill ring. vi users know the difference between normal mode and insert mode. Web browser users know about URLs (or, at least, addresses) and links. Pantry is no different, as it comes with its own set of terminology and its own paradigm, as you will learn about in this chapter.

A client-server model

Pantry is designed to work entirely from the command line. This gives you full access to the power of your shell as you work with Pantry. However, Pantry needs to deal with lots of data. It would take too long to load all this data into memory every time you execute a Pantry command.

The solution is a client-server model. When you use Pantry, you'll start a server which runs in the background. This server will hold all the data you're working with. To interact with the server from the command line, you'll use a client program. When you're done with the server, you can shut it down--or you can just leave it running. It's up to you.

Pantry comes with two binary programs, with **pantryd** being the server and **pantry** being the client. Generally you will only run one server at a time. The server and client communicate with each other using a UNIX domain socket [http://en.wikipedia.org/wiki/Unix_domain_sockets], which is a common means of interprocess communication on Unix. By default, this socket is a file at `$HOME/.pantrySocket`. If you want to use a different location for your socket file, you can do so by setting the environment variable PANTRY_SOCKET before you start **pantryd**, and you will also need to be sure the PANTRY_SOCKET variable is set each time you run **pantry**. Specify the entire path to the socket, including the filename itself (it need not be named `.pantrySocket`). By using the PANTRY_SOCKET variable, you can run more than one **pantryd** server simultaneously.[1]

Foods

You will work with Pantry by manipulating what we will call foods. Each food in Pantry has various pieces of data associated with it, called traits, nutrients, and available units.

Traits

A food's traits is a set of twelve strings. Each of these strings indicates something interesting about the food. For instance, four of these traits are `name`, `date`, `quantity`, and `unit`. For a particular food, these might be equal to `Apples, raw, with skin`, `Wednesday` (to indicate the date that you ate it) `2` and `large (3-1/4" dia)` to indicate that you ate two large apples. Of course you can change a food's traits. For instance, you can rename any food whenever you want, and you can change its quantity and unit traits to indicate how much you ate (or just to see the nutrient breakdown for that particular quantity and unit.)

Twelve food traits

`name` This is the name of the food, such as `Apples, raw, with skin`.

[1] You might need to use PANTRY_SOCKET if the default socket location does not work for you because, for example, you don't have permission to create files in your home directory or if your home directory is on a filesystem that does not allow creation of sockets. Otherwise, I can't imagine why most folks would want to run more than one server simultaneously, though this feature does come in handy when I am testing new versions of Pantry and preparing them for release.

`group` — You can assign whatever you like to this trait to group foods. The foods in the `starter_database` file already have group traits corresponding to the food groups used by USDA; for instance, `Apples, raw, with skin` has a group trait of `Fruits and Fruit Juices`.

`quantity` — This is a number. You can use an integer, like `2`; a decimal, like `2.5`; a fraction, like `1/2`; or a mixed number, like `2 1/2`.

`unit` — This is a string. It must be equal to one of the food's available units. Every food has `g`, `oz`, and `lb` as available units; you can also assign additional available units to a food, as we will discuss later. Later we will also see how you can find out what a food's available units are.

`date` — You can assign whatever you wish to this trait to indicate when you ate a food, if you wish. This does not need to be formatted any particular way, so you can use `2008-05-18`, `Sunday`, or `the day before yesterday` if you want.

`meal` — A string indicating what meal you ate a food at, such as `Lunch` or `midday snack`.

`comment` — Any comments you may wish to add.

`order` — Sometimes you may want your foods to be sorted in a particular order when you print reports; to do that you can assign appropriate values to this trait, such as `a` for one food and `b` for the next food. As we will see later, Pantry can also assign values to this trait for you if you wish.

`refuse` — A description of the refuse of a food--that is, the part you don't eat, like bones or shells. For `Apples, raw, with skin`, this is `Core and stem`.

`percent-refuse` — The percent of the food that is refuse. As we will see later Pantry can use the value of this trait to make it easier for you to keep track of food quantities. As with the quantity trait, this can be an integer, decimal value, fraction, or mixed number. For `Apples, raw, with skin`, this is `10`, which is roughly the percentage of an apple's weight that is core and stem.

`yield` — Later you will learn how to create your own recipes in Pantry. This trait indicates the total yield of a recipe, in grams.

`instructions` — For a recipe, this may contain any preparation instructions you may wish to add.

Nutrients

A food's nutrients is a set of pairs of strings and numbers. These indicate the nutrient breakdown of a food. For example, the food with the name trait `Apples, raw, with skin` whose quantity trait is `1` and whose unit trait is `large (3-1/4" dia)` has 100 nutrients total. Some of the more interesting ones are `Calories` which is `116`; `Calcium, mg` which is 13, and `Dietary Fiber, g` which is `5`. A food's nutrients adjust automatically as you change its quantity or unit traits.

Available units

A food's available units indicate various possible quantities of a food, and how much each given quantity weighs in grams. For example, the food whose name trait is `Apples, raw, with skin` has ten available units total. Among these are `large (3 1/4" dia)`, which is 223g, and `cup slices`, which is 109g. Every food always has `g`, `oz`, and `lb` as available units.

Collections and databases

A collection holds foods. Every food must be in a collection. You can use collections to group foods in a way that is convenient to you--for example, by day of the week, or by foods you eat frequently. A database holds one or more collections. In Pantry you can only work with one database at a time, though you can work with more than one collection as long as they are in the same database. You can save databases to disk so you can load them again later.

Chapter 2. Manipulating foods

Now that you know some essential Pantry terminology, you're ready to jump in and do something interesting. In this chapter you'll learn how to use Pantry to find nutrition information about foods.

Opening files and getting server status

As you learned in the previous chapter, Pantry employs a client-server architecture. So the first thing you must do when working with Pantry is start a server. To start a server, simply type **`pantryd`** at a command prompt:

Example 2.1. Starting the Pantry server

```
$ pantryd
```

This starts up a Pantry server in the background and returns you to your prompt. This server will keep running, even after you logout of the shell from which you started the server.[1] If you're a skeptic and you don't believe that you actually have a program running in the background, you can verify that a Pantry server is running by using your **ps** command.

Now to interact with the server you'll use the **pantry** command. First let's load a database. Pantry comes with the `starter_database` file. It contains over 7,000 foods from the USDA's National Nutrient Database for Standard Reference [http://www.ars.usda.gov/Services/docs.htm?docid=8964]. You'll find this file at `$PREFIX/share/pantry/data/starter_database`, where $PREFIX is the prefix you used when you ran the configure script. (This defaults to `/usr/local`.) So to open the file run a command similar to this. Just make an appropriate substitution for the location of your `starter_database` file:

Example 2.2. Opening a database

```
$ pantry --open data/starter_database
```

As with the previous command, you're returned to your prompt without a message. Pantry generally is quite terse--if it has nothing surprising to say, or if you have not asked it to say anything, then it stays quiet. If you're wondering if that last command really did anything, you can ask the server to report its status:

Example 2.3. Getting server status

```
$ pantry --status
Current filename: /home/massysett/pantry-git/data/starter_database
Collections:
  7413 master
```

This shows that the database you just opened has a single collection, called `master`, with 7413 foods.

The `name`, `info`, and `blank` reports

Next let's learn how to print reports, which are Pantry's way of telling you what you want to know about foods. Most invocations of the Pantry command take the form **`pantry [options] COLLECTION...`**

[1] As we will discuss later, to terminate the server, run **`pantry --quit`**.

When you enter a **pantry** command, you will usually specify one or more collections as non-option arguments[2]. For example, as we saw above, the `starter_database` file comes with one collection, named `master`.

In order to learn exactly what foods are in the `master` collection, you can print reports about the foods. For example, the simplest report is the `name` report, which prints the name trait of every food in the buffer:

Example 2.4. The `name` report

```
$ pantry --print name master | head
Beans, cranberry (roman), mature seeds, canned
Beans, cranberry (roman), mature seeds, cooked, boiled, without salt
Beans, cranberry (roman), mature seeds, raw
Beans, black turtle soup, mature seeds, canned
Beans, black turtle soup, mature seeds, cooked, boiled, without salt
Beans, black turtle soup, mature seeds, raw
Beans, black, mature seeds, cooked, boiled, without salt
Beans, black, mature seeds, raw
Beans, baked, canned, with pork and tomato sauce
Beans, baked, canned, with pork and sweet sauce
```

Thus you use the `--print` option to print reports. The `--print` option takes a single argument, which is the name of the report--here, `name`. Finally you specify the collection or collections that has the foods you're interested in--here, `master`. Pantry first copies every food from the `master` collection into the buffer. It then prints a name report for every one of those foods. That's a lot of foods--over 7,000 of them, so in the manual I've piped the output to **head** to save some space. After your pantry command is done, the foods that were copied to the buffer are gone, and the foods in the `master` collection are unchanged.

To know more than just the name of a food, use the `info` report. It prints information about most of a food's traits. It does not print information about the food's `name` trait (for that, use the `name` report) or about some of the traits that are related only to foods that are recipes (we'll discuss reports applicable to recipes later.) To save space, the `info` report does not print information about a trait if it is blank. You can print more than one report at a time simply by specifying multiple `--print` options.

Example 2.5. The `info` report

```
$ pantry --print name --print info master | head
Beans, cranberry (roman), mature seeds, canned
100 g (100g)
Group: Legumes and Legume Products
Beans, cranberry (roman), mature seeds, cooked, boiled, without salt
100 g (100g)
Group: Legumes and Legume Products
Beans, cranberry (roman), mature seeds, raw
100 g (100g)
Group: Legumes and Legume Products
Beans, black turtle soup, mature seeds, canned
```

For every food in the buffer, Pantry first printed a `name` report and then printed an `info` report. As you can see, some whitespace would help make this more readable. To get whitespace use the `blank` report, which prints a blank line:

[2]When I say non-option arguments, it's to distinguish these arguments from the arguments with leading single or double dashes.

Example 2.6. The `blank` report

```
$ pantry --print name --print info \
--print blank master | head
Beans, cranberry (roman), mature seeds, canned
100 g (100g)
Group: Legumes and Legume Products

Beans, cranberry (roman), mature seeds, cooked, boiled, without salt
100 g (100g)
Group: Legumes and Legume Products

Beans, cranberry (roman), mature seeds, raw
100 g (100g)
```

All the reports we have seen so far are printed once for every food in the buffer. We'll call these reports food reports. On the other hand, some reports are printed just once for the entire buffer, no matter how many foods are in the buffer. We'll call these reports summary reports because they summarize information for the entire buffer. One summary report is the `groups` report, which tells you information about the group trait of every food in the buffer:

Example 2.7. The `groups` report

```
$ pantry --print groups master
Group name                                          Food count
-----------------------------------------------------------
Baby Foods                                          314
Baked Products                                      489
Beef Products                                       485
Beverages                                           262
Breakfast Cereals                                   440
Cereal Grains and Pasta                             180
Dairy and Egg Products                              222
Ethnic Foods                                        153
Fast Foods                                          354
Fats and Oils                                       220
Finfish and Shellfish Products                      255
Fruits and Fruit Juices                             321
Lamb, Veal, and Game Products                       345
Legumes and Legume Products                         359
Meals, Entrees, and Sidedishes                      123
Nut and Seed Products                               128
Pork Products                                       319
Poultry Products                                    371
Sausages and Luncheon Meats                         233
Snacks                                              139
Soups, Sauces, and Gravies                          497
Spices and Herbs                                    61
Sweets                                              338
Vegetables and Vegetable Products                   805
-----------------------------------------------------------
  Total                                             7413
```

Selecting foods with the `--search` option

As you learned in the previous section, the information in your reports in Pantry depends on what foods are in the buffer, and when you specify a collection name as an argument to the **pantry** command, Pantry copies all the foods from that collection into the buffer. So every report we saw in the previous section pertained to all the foods in the `master` collection in the `starter_database` file--that's over 7,000 foods. Often you'll only be interested in a fraction of the foods in a particular collection. That's where the `--search` option comes in.

With the `--search` option, you specify a particular trait, such as name, date, or group, and a pattern you wish to match for that particular trait. As we learned before, Pantry first copies every food from the each collection you specify into the buffer. However, when you specify `--search` options, Pantry then eliminates every food whose trait does not match the pattern you specified. An example will help:

Example 2.8. Using `--search name`

```
$ pantry --search name "Apples, raw, with skin" \
--print name --print info master
Apples, raw, with skin
100 g (100g)
Group: Fruits and Fruit Juices
Refuse: 10 percent Core and stem
```

You have to quote the second argument to the `--search` option if, as here, it contains characters that the shell gives special meaning to, such as spaces. Here [http://www.mpi-inf.mpg.de/~uwe/lehre/unixffb/quoting-guide.html] is an excellent guide that explains why and when quoting is necessary.

As with the `--print` option, you may specify multiple `--search` options in a single command. The effect of multiple `--search` options is cumulative; that is, a food must match all of the `--search` options specified, or Pantry will remove it from the buffer.

Example 2.9. Using multiple `--search` options

```
$ pantry --search name Apples --print name master
Applesauce, canned, sweetened, with salt
Applesauce, canned, unsweetened, with added ascorbic acid
Applesauce, canned, sweetened, without salt (includes USDA commodity)
Applesauce, canned, unsweetened, without added ascorbic acid (includes USDA commod
Apples, frozen, unsweetened, heated
Apples, frozen, unsweetened, unheated
Apples, dried, sulfured, stewed, with added sugar
Apples, dried, sulfured, stewed, without added sugar
Apples, dried, sulfured, uncooked
Apples, dehydrated (low moisture), sulfured, stewed
Apples, dehydrated (low moisture), sulfured, uncooked
Apples, canned, sweetened, sliced, drained, heated
Apples, canned, sweetened, sliced, drained, unheated
Apples, raw, without skin, cooked, microwave
Apples, raw, without skin, cooked, boiled
Apples, raw, without skin
Apples, raw, with skin

$ pantry --search name Apples \
--search name skin --print name master
Apples, raw, with skin
Apples, raw, without skin
Apples, raw, without skin, cooked, boiled
Apples, raw, without skin, cooked, microwave
```

You can search using any trait you wish, not just the name trait. For example, you can show all foods with the word `Fruit` in their group trait:

Example 2.10. Using the `group` trait in a search

```
$ pantry --search group Fruit --print name \
--print info --print blank master | head
Pineapple, canned, juice pack, solids and liquids
100 g (100g)
Group: Fruits and Fruit Juices

Pineapple, canned, water pack, solids and liquids
100 g (100g)
Group: Fruits and Fruit Juices

Pineapple, raw, all varieties
100 g (100g)
```

As with many things Unix, by default the `--search` option is case sensitive. To make it case insensitive, use the `--ignore-case` option. This makes the pattern to every `--search` option case insensitive.

Example 2.11. The `--ignore-case` option

```
$ pantry --search name papaya \
--print name master
Babyfood, fruit, guava and papaya with tapioca, strained
Babyfood, fruit, papaya and applesauce with tapioca, strained
Fruit salad, (pineapple and papaya and banana and guava), tropical, canned, heavy

$ pantry --ignore-case \
--search name papaya \
--print name master
Babyfood, fruit, guava and papaya with tapioca, strained
Babyfood, fruit, papaya and applesauce with tapioca, strained
Papayas, raw
Papaya nectar, canned
Fruit salad, (pineapple and papaya and banana and guava), tropical, canned, heavy
```

Also, by default the search patterns are regular expressions [http://www.regular-expressions.info].[3] If a regular expression matches any part of the trait, then the food matches. This can have mildly unexpected results. Sometimes you may wish to use anchors [http://www.regular-expressions.info/anchors.html]. In addition, there is an `--exact-match` option which turns off regular expressions. When you use exact match, all search patterns must exactly match the search patterns given.

Example 2.12. Patterns, anchors, and `--exact-match`

```
$ pantry --search name "Apples, raw, without skin" \
--print name master
Apples, raw, without skin
Apples, raw, without skin, cooked, boiled
Apples, raw, without skin, cooked, microwave

$ pantry --search name "Apples, raw, without skin$" \
--print name master
Apples, raw, without skin

$ pantry --exact-match \
--search name "Apples, raw, without skin" \
--print name master
Apples, raw, without skin
```

The `units` report, and changing foods

Now you know how to search for foods and print information about them. This is often useful all by itself, but often you will want to change foods. To change a trait, just use the `--change` option:

[3] Pantry uses the regular expression routines that come with the standard UNIX C library. Pantry uses "extended" regular expressions; typically they are similar to what your egrep command uses. These regular expressions are less versatile than those Perl uses, but they are usually good enough and they are available on any Unix system.

Example 2.13. Changing traits

```
$ pantry --ignore-case --search name papaya \
--change group "Tropical Fruits" \
--print name --print info --print blank \
master
Babyfood, fruit, guava and papaya with tapioca, strained
100 g (100g)
Group: Tropical Fruits

Babyfood, fruit, papaya and applesauce with tapioca, strained
100 g (100g)
Group: Tropical Fruits

Papayas, raw
100 g (100g)
Group: Tropical Fruits
Refuse: 33 percent Seeds and skin

Papaya nectar, canned
100 g (100g)
Group: Tropical Fruits

Fruit salad, (pineapple and papaya and banana and guava), tropical, canned, heavy
100 g (100g)
Group: Tropical Fruits
```

As you already know, when you execute a pantry command, Pantry first copies all foods from every collection you specify into a buffer. Pantry then removes from the buffer all foods that do not match any `--search` optionis you specified. Next is where the `--change` option comes in. Pantry changes every food in the buffer using any `--change` options you have specified. After that, Pantry prints any reports you have requested with the `--print` option.

Remember that Pantry first copies foods from the collections into the buffer, so when you use the change option, the original foods that are in the collection are not changed. In the last example we changed the group trait of some foods that have `papaya` in their name trait, but as this shows, this change did not affect the foods in the `master` collection:

Example 2.14. Changes only affect foods in the buffer

```
$ pantry --ignore-case --search name papaya \
--print name --print info --print blank \
master
Babyfood, fruit, guava and papaya with tapioca, strained
100 g (100g)
Group: Baby Foods

Babyfood, fruit, papaya and applesauce with tapioca, strained
100 g (100g)
Group: Baby Foods

Papayas, raw
100 g (100g)
Group: Fruits and Fruit Juices
Refuse: 33 percent Seeds and skin

Papaya nectar, canned
100 g (100g)
Group: Fruits and Fruit Juices

Fruit salad, (pineapple and papaya and banana and guava), tropical, canned, heavy
100 g (100g)
Group: Fruits and Fruit Juices
```

To make changes permanent, you can use the `--add`, `--edit`, and `--delete` options, which we will talk about later.

As we discussed earlier, every food in Pantry has twelve traits. Of these twelve, you can assign whatever text you want to eight: `name`, `group`, `date`, `meal`, `comment`, `order`, and `refuse`. Of course you might be less confused if you, say, refrain from using the refuse trait to store information about when you ate a food, but that is up to you. You are not constrained to using a particular format for the date trait; later on, when we discuss the `--key` option, we'll learn of some considerations you might want to take into account when assigning information to this trait.

The `quantity` trait is a number. When you assign values to it using the `--change` option, you must use a value that evaluates to a number. You can use whole numbers (such as `2`), decimals (such as `2.5`), fractions (such as `2/3`) or mixed numbers (such as `1 2/3`). If you try to use anything else for the quantity trait, Pantry will give you an error message and quit.

Example 2.15. Changing the `quantity` trait

```
$ pantry --search name "Apples, raw, with skin" \
--change quantity "2 1/2" \
--print name --print info master
Apples, raw, with skin
2 1/2 g (2.5g)
Group: Fruits and Fruit Juices
Refuse: 10 percent Core and stem
```

Entering `0` for a food's quantity trait does not delete a food; to do that, use the `--delete` option which we will talk about later.

The `percent-refuse` trait is similar to the `quantity` trait in that it is also a number. However, the `percent-refuse` trait has one additional requirement: it must be between `0` and `100`, inclusive, because a food can't have less than zero percent refuse or more than 100 percent refuse.

As we discussed much earlier, each food in Pantry may have a number of available units. Knowing what these are will come in handy later when you wish to change the units of a food, but for now let's just see what a `units` report looks like.

Example 2.16. Units report

```
$ pantry --ignore-case --search name papaya \
--print name --print units --print blank \
master
Babyfood, fruit, guava and papaya with tapioca, strained
   jar (113g)

Babyfood, fruit, papaya and applesauce with tapioca, strained
   jar (113g)

Papayas, raw
   cup, cubes (140g)
   cup, mashed (230g)
   large (5-3/4" long x 3-1/4" dia) (380g)
   medium (5-1/8" long x 3" dia) (304g)
   small (4-1/2" long x 2-3/4" dia) (152g)

Papaya nectar, canned
   cup (250g)
   fl oz (31g)

Fruit salad, (pineapple and papaya and banana and guava), tropical, canned, heavy
   cup (257g)
```

Every food has `g`, `oz`, and `lb` available as units, so the units report does not show these to save some space.

The `unit` trait must be set to be equal to one of the food's available units. When you are changing a food's unit, the second argument to the `--change` option is a regular expression that must match exactly one of the food's available units. If the regular expression does not match exactly one of the food's available units, Pantry will give you an error message and quit.

Example 2.17. Changing the `unit` trait

```
$ pantry --search name "Papayas, raw" \
--change quantity 1 \
--change unit large \
--print name --print info master
Papayas, raw
1 large (5-3/4" long x 3-1/4" dia) (380g)
Group: Fruits and Fruit Juices
Refuse: 33 percent Seeds and skin
```

As always with the `--change` options, using `--change` unit will change every food in the buffer so that its unit matches the pattern you give. For example, take the following output from a `units` report:

Example 2.18. Papaya units reports

```
$ pantry --ignore-case --search name papaya \
--print name --print units --print blank \
master
Babyfood, fruit, guava and papaya with tapioca, strained
   jar (113g)

Babyfood, fruit, papaya and applesauce with tapioca, strained
   jar (113g)

Papayas, raw
   cup, cubes (140g)
   cup, mashed (230g)
   large (5-3/4" long x 3-1/4" dia) (380g)
   medium (5-1/8" long x 3" dia) (304g)
   small (4-1/2" long x 2-3/4" dia) (152g)

Papaya nectar, canned
   cup (250g)
   fl oz (31g)

Fruit salad, (pineapple and papaya and banana and guava), tropical, canned, heavy
   cup (257g)
```

Remember that every food has `g`, `oz`, and `lb` as available units, so I could have appended **`--change unit "lb"`** to the previous command. However, had I appended **`--change unit small`** to the previous command, **pantry** would have given me an error message and quit because only one of the foods in the buffer had an available unit that matched that pattern.

Let's say I want to change a papaya's unit to `g` (for grams). What if I wanted to run **`pantry --search name "Papayas, raw" --change unit g master`**? That would give me an error message because `g` matches both `g` and `large (5-3/4" long x 3-1/4" dia)`. One solution to this is to use anchors by specifying **`--change unit '^g$'`** instead.[4] Another is to use the `--exact-match` option. Because the second argument to the `--change unit` option is a regular expression, it respects the `--exact-match` and `--ignore-case` options just as the `--search` option does. Thus, **`pantry --exact-match --search name "Papayas, raw" --change unit g master`** works perfectly.

Making changes permanent

As we've been pointing out, when you make changes to foods using the `--change` option, these changes are not permanent. This is because those changes are to foods that are in the buffer, which is Pantry's temporary holding place for foods. You get a new buffer every time you run a pantry command.

There are three ways to make your changes permanent. The one you'll use the most is the `--add` option. This takes the entire buffer and adds its contents to the specified collection:

[4] I do this with some frequency, so to make it a little easier to type I have **`alias -g G='\^g\$'`** in a file that defines Pantry shell functions, parameters, and aliases for me. This will only work in **zsh**.

Example 2.19. Adding foods to a collection

```
$ pantry --search name "Apples, raw, with skin" \
--change quantity 2 \
--change unit medium \
--change date 2008-06-22 \
--change meal Lunch \
--add diary \
master

$ pantry --print name --print info diary
Apples, raw, with skin
2 medium (3" dia) (364g)
Group: Fruits and Fruit Juices
Date: 2008-06-22
Meal: Lunch
Refuse: 10 percent Core and stem
```

The `--add` option is the easiest way to keep a food diary. You simply search for foods in the `master` collection, change their traits as appropriate (such as their `quantity`, `date`, and `meal` traits) and then add them to another collection. I typically add all my diary foods to one collection, `diary`, but you can use multiple collections for this purpose--such as one collection for each day, week, or whatever. You can even add a food to more than one collection at a time; just use multiple `--add` options, one for each collection.

Once you have added a food to a collection with `--add,` that food is independent of the collection it came from. That is, in the previous example, if you then make changes to the apple you just added to diary, this will not affect the food in the `master` collection.

Thus the `--add` option works by copying foods from the buffer into a collection. Sometimes though you will want to change a food that is already in a collection, without copying it. For example, let's say that in the previous example I made a mistake. I really wanted to indicate that I ate one medium apple, not two. The `--edit` option will help you here.

Example 2.20. Using the `--edit` option

```
$ pantry --search name "Apples, raw, with skin" \
--search date 2008-06-22 \
--search meal Lunch \
--change quantity 1 \
--edit \
diary

$ pantry --search date 2008-06-22 \
--print name --print info --print blank \
diary
Apples, raw, with skin
1 medium (3" dia) (182g)
Group: Fruits and Fruit Juices
Date: 2008-06-22
Meal: Lunch
Refuse: 10 percent Core and stem
```

With the `--edit` option, Pantry copies foods from the source collections and makes the changes indicated using the --change option, as it always does. However, when you use `--edit,` Pantry will make changes

to the foods that are inside the original collections, as well as making changes to the foods that are in the buffer.

Or, let's say that I decided that I didn't really eat an apple after all. Then I can use the `--delete` option:

Example 2.21. Using the `--delete` option

```
$ pantry --search date 2008-06-22 \
--search name Apples \
--print name \
--delete diary
Apples, raw, with skin

$ pantry --status
Current filename: /home/massysett/pantry-git/data/starter_database
Collections:
     0 diary (changed)
  7413 master
```

As you can see, the diary collection now has no foods because you deleted the single food that was in the collection.

With the `--delete,` option, after copying foods into the buffer and making any changes you specify, Pantry then deletes the foods from the original collections. The foods are still present in the buffer, which is why you can still `--print` them.

Reports about nutrients

There are two reports to tell you about the nutrient content of foods. First we will discuss the `goals` report. It allows you to specify the nutrients you wish to see in a report:

Example 2.22. The `goals` report

```
$ pantry --search name "Apples, raw, with skin" \
--print name --print info --print goals \
--goal Calories 2600 \
--goal "Total Fat, g" 43 \
--goal "Protein, g" 162 \
--goal "Total Carbohydrate, g" 390 \
--goal "Dietary Fiber, g" 0 \
master
Apples, raw, with skin
100 g (100g)
Group: Fruits and Fruit Juices
Refuse: 10 percent Core and stem
Nutrient name                          Amount %G    %T
-------------------------------------------------------------
Calories                               52     2     100
Total Fat, g                           0      0     100
Protein, g                             0      0     100
Total Carbohydrate, g                  14     4     100
Dietary Fiber, g                       2            100
```

As the example illustrates, you specify the nutrients you wish to see in the goals report by using multiple `--goal` options, and the nutrients are shown in the order in which you asked for them with the `--goal`

options. The first argument to each `--goal` option specifies the nutrient you wish to see. Unlike many other things in Pantry, this argument is not a regular expression--instead, for a nutrient to be printed in the goals report, its name must exactly match one of the nutrients specified with a `--goal` option. The `--goal` option is always case sensitive, even if you have specified the `--ignore-case` option.

The second argument to each `--goal` option is a goal. For instance, here I have specified that it is my goal to eat 2600 calories, 43 grams of fat, and so on. These goals can be per day, per month, per meal...whatever you want. If you don't have a goal for a particular nutrient, just specify `0`. (You do, however, have to specify a goal, even if it is zero.)

With that, you can probably figure out what all the columns in the goals report mean. The first is simply the name of the nutrient. The second is the amount of the nutrient present. The third column is the ratio of the second column to the goal you specified, expressed as a percentage.

The fourth column shows this nutrient's percentage of the total amount of that nutrient in the buffer. This makes more sense with an example. Suppose I am a fruititarian:

Example 2.23. My fruititarian foods

```
$ pantry --search name 'Apples, raw, with skin' \
--change quantity 3 \
--change unit large \
--change date 2008-06-12 \
--change meal Breakfast \
--add diary \
master

$ pantry --search name 'Bananas, raw' \
--change quantity 1 \
--change unit medium \
--change date 2008-06-12 \
--change meal Lunch \
--add diary \
master

$ pantry --search name 'Papayas, raw' \
--change quantity 2 \
--change unit large \
--change date 2008-06-12 \
--change meal Dinner \
--add diary \
master
```

Example 2.24. My fruititarian report

```
$ pantry --search date 2008-06-12 \
--print name --print info --print goals \
--print blank \
--goal Calories 2600 \
--goal "Total Fat, g" 43 \
--goal "Protein, g" 162 \
--goal "Total Carbohydrate, g" 390 \
--goal "Dietary Fiber, g" 0 \
diary
Apples, raw, with skin
3 large (3-1/4" dia) (669g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Breakfast
Refuse: 10 percent Core and stem
Nutrient name                          Amount %G    %T
-------------------------------------------------------
Calories                               348    13    46
Total Fat, g                           1      3     44
Protein, g                             2      1     23
Total Carbohydrate, g                  92     24    48
Dietary Fiber, g                       16           49

Bananas, raw
1 medium (7" to 7-7/8" long) (118g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Lunch
Refuse: 36 percent Skin
Nutrient name                          Amount %G    %T
-------------------------------------------------------
Calories                               105    4     14
Total Fat, g                           0      1     15
Protein, g                             1      1     17
Total Carbohydrate, g                  27     7     14
Dietary Fiber, g                       3            9

Papayas, raw
2 large (5-3/4" long x 3-1/4" dia) (760g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Dinner
Refuse: 33 percent Seeds and skin
Nutrient name                          Amount %G    %T
-------------------------------------------------------
Calories                               296    11    40
Total Fat, g                           1      2     41
Protein, g                             5      3     61
Total Carbohydrate, g                  75     19    38
Dietary Fiber, g                       14           42
```

The `goals` report tells you only about the nutrients that you ask for. To know about all the nutrients in a food, use the `nuts` report instead. Because the people at USDA are so industrious, the foods in the `starter_database` file have lots of nutrients. Let's see some of them for the apple. Of course when you run this on your computer you can omit the **`| head`**, but I did that here to save some space in the manual. The columns in the `nuts` report mean the same as the corresponding columns in the goals report; however, unlike the `goals` report, you do not need to enter a goal for a nutrient in order for the nutrient to be shown in the nuts report; instead, all of a food's nutrients are shown.

Example 2.25. The `nuts` report

```
$ pantry --search name "Apples, raw, with skin" \
--print name --print nuts master | head
Apples, raw, with skin
Nutrient name                              Amount %G    %T
-------------------------------------------------------------
10:0, g                                    0
12:0, g                                    0
14:0, g                                    0            100
16:0, g                                    0            100
16:1 undifferentiated, g                   0
18:0, g                                    0            100
18:1 undifferentiated, g                   0            100
```

The `--goals` and `--nuts` report tell you about each food in the buffer. To get information about the total nutrient content of a buffer, use the `Goals` and `Nuts` summary reports.

Example 2.26. Using the `Goals` and `Nuts` reports

```
$ pantry --search date 2008-06-12 \
--print Goals \
--goal Calories 2600 \
--goal "Total Fat, g" 43 \
--goal "Protein, g" 162 \
--goal "Total Carbohydrate, g" 390 \
--goal "Dietary Fiber, g" 0 \
diary
Sum of nutrients with a goal:
Nutrient name                              Amount %G
-------------------------------------------------------
Calories                                   749    29
Total Fat, g                               3      6
Protein, g                                 8      5
Total Carbohydrate, g                      194    50
Dietary Fiber, g                           33

$ pantry --search date 2008-06-12 \
--print Nuts diary | head
Sum of all nutrients:
Nutrient name                              Amount %G
-------------------------------------------------------
10:0, g                                    0
12:0, g                                    0
14:0, g                                    0
16:0, g                                    1
16:1 undifferentiated, g                   0
18:0, g                                    0
18:1 undifferentiated, g                   0
```

Sorting reports, and using `--auto-order`

For all practical purposes, Pantry stores foods in collections in no particular order. Thus, when you use the `--print` option to print reports, you don't know what order your foods will come out in. That can make it hard to find the information you're looking for.

That's why Pantry has options so it will sort foods in an order you specify. You sort foods by adding keys. A key is a particular trait that you wish to use to sort foods. For example:

Example 2.27. A basic sorting example

```
$ pantry --key name \
--print name --print info --print blank \
diary
Apples, raw, with skin
3 large (3-1/4" dia) (669g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Breakfast
Refuse: 10 percent Core and stem

Bananas, raw
1 medium (7" to 7-7/8" long) (118g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Lunch
Refuse: 36 percent Skin

Papayas, raw
2 large (5-3/4" long x 3-1/4" dia) (760g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Dinner
Refuse: 33 percent Seeds and skin
```

If you want Pantry to sort in descending rather than ascending order,[5] use a capital letter to specify the name of the trait:

[5]Pantry sorts in what has sometimes been called ASCIIbetical order. Pantry knows nothing about multibyte characters. See Appendix A, *Pantry limitations*.

Example 2.28. Sorting in descending order

```
$ pantry --key Name \
--print name --print info --print blank \
diary
Papayas, raw
2 large (5-3/4" long x 3-1/4" dia) (760g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Dinner
Refuse: 33 percent Seeds and skin

Bananas, raw
1 medium (7" to 7-7/8" long) (118g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Lunch
Refuse: 36 percent Skin

Apples, raw, with skin
3 large (3-1/4" dia) (669g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Breakfast
Refuse: 10 percent Core and stem
```

You can also specify more than one `--key`. Then Pantry will first sort foods using the first key, then by the second key, etc. For instance, here Pantry will sort foods first by their `date` traits. Foods with identical `date` traits will be sorted by `name`.

Example 2.29. Specifying more than one `--key`

```
$ pantry --search name \
"Snacks, popcorn, microwave, low fat" \
--change quantity 2 \
--change unit oz \
--change date 2008-06-20 \
--change meal Lunch \
--add diary master

$ pantry --search name \
"Carbonated beverage, cola, contains caffeine" \
--change quantity 1 \
--change unit can \
--change date 2008-06-20 \
--change meal Lunch \
--add diary master

$ pantry --key date --key name \
--print name --print info --print blank \
diary
Apples, raw, with skin
3 large (3-1/4" dia) (669g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Breakfast
Refuse: 10 percent Core and stem

Bananas, raw
1 medium (7" to 7-7/8" long) (118g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Lunch
Refuse: 36 percent Skin

Papayas, raw
2 large (5-3/4" long x 3-1/4" dia) (760g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Dinner
Refuse: 33 percent Seeds and skin

Carbonated beverage, cola, contains caffeine
1 can 12 fl oz (368g)
Group: Beverages
Date: 2008-06-20
Meal: Lunch

Snacks, popcorn, microwave, low fat
2 oz (56.7g)
Group: Snacks
Date: 2008-06-20
Meal: Lunch
```

You might want to sort your foods in a particular order rather than alphabetical order. For instance, you might want to see foods sorted by meal. This would yield `Breakfast`, `Dinner`, `Lunch` if sorted alphabetically. To fix this, use the `--list` option. This allows you to list values for a particular trait. Foods will then be sorted in the order that you specified. (All foods having a trait whose value is not listed with the `--list` option will be sorted alphabetically, after foods whose trait values were specified with `--list`.)

Example 2.30. Using the `--list` option to sort foods

```
$ pantry --search date 2008-06-12 \
--list meal Breakfast \
--list meal Lunch \
--list meal Dinner \
--key meal \
--print name --print info \
--print blank diary
Apples, raw, with skin
3 large (3-1/4" dia) (669g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Breakfast
Refuse: 10 percent Core and stem

Bananas, raw
1 medium (7" to 7-7/8" long) (118g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Lunch
Refuse: 36 percent Skin

Papayas, raw
2 large (5-3/4" long x 3-1/4" dia) (760g)
Group: Fruits and Fruit Juices
Date: 2008-06-12
Meal: Dinner
Refuse: 33 percent Seeds and skin
```

The `--list` option is always case sensitive; it is not affected by the `--ignore-case` option. The second argument to the `--list` option is just a plain string, not a regular expression pattern.

You may find that you want to add foods to a collection in a particular order and then see them reported in that order. Because Pantry stores foods in a manner that is for all practical purposes unordered, just adding foods to a collection in a particular order does not ensure they will be shown in that order in reports. Instead, that's what the `--auto-order` option is for. If you use it when adding a food to a collection, Pantry will automatically scan the collection for foods with identical `date` and `meal` traits. Pantry will then set the order trait of the food that is added to the collection so that the new food's `order` trait will come after all the other foods with identical `date` and `meal` traits. Then you can sort foods using the `order` trait to see them in the order in which you added them.

Phew! That's a long explanation. An example will help:

Example 2.31. Using the `--auto-order` option

```
$ pantry --search name "McDONALD'S, French Fries" \
--change quantity 1 \
--change unit large \
--change date 2008-06-21 \
--change meal Lunch \
--add diary --auto-order \
master

$ pantry --search name "McDONALD'S, Double Cheeseburger" \
--change quantity 1 \
--change unit item \
--change date 2008-06-21 \
--change meal Lunch \
--add diary --auto-order \
master

$ pantry --search date 2008-06-21 \
--search meal Lunch \
--key order \
--print name --print info --print blank \
diary
McDONALD'S, French Fries
1 large serving (154g)
Group: Fast Foods
Date: 2008-06-21
Meal: Lunch
Order: 0010

McDONALD'S, Double Cheeseburger
1 item (173g)
Group: Fast Foods
Date: 2008-06-21
Meal: Lunch
Order: 0020
```

`--auto-order` does not require that you change either the `date` or `meal` traits of foods; it will work fine if those traits are left blank (all the foods in the `starter_database` file have blank date and meal traits.)

Approximations with the `--refuse` and `--by-nut` options

"There's good, and then there's good enough," I like to say. Pantry has two options that can help you when you are looking for approximations.

First the `--refuse` option. So far we have paid little attention to the `refuse` trait, which specifies the typical percentage of a food that is inedible. If you use this option, Pantry will reduce the quanitity of each food in the buffer by each food's percent-refuse trait. For example, apples range in size from tiny (about 6 oz, say) to enormous (over a pound!) If I am eating an apple, there are various ways I can use Pantry to

find out the apple's nutrient content. First, I can look at it, guess its size, and then use the units report to find the appropriate available unit for `Apples, raw, with skin`. This requires that I make a good guess, but I am quite bad at guessing these things.

I could weigh the apple. The problem here is that the foods in the `starter_database` file have nutrient amounts for a given edible portion of food. These amounts don't include the core, stem, and seeds of the apple. So for the best accuracy I would have to cut up the apple and remove the core, stem, and seeds before weighing it.

Another alternative is the `--refuse` option. This way I can weigh the apple whole. Pantry will then deduct the appropriate amount from the quantity.

Example 2.32. Using the `--refuse` option

```
$ pantry --search name "Apples, raw, with skin" \
--refuse \
--change quantity 395 \
--change unit '^g$' \
--print name --print info master
Apples, raw, with skin
355.500000000000000 g (355.5g)
Group: Fruits and Fruit Juices
Refuse: 10 percent Core and stem
```

Of course you can tack on options like `--print goals` and `--goal` in order to find more useful information about the apple.

There's no reason to use the `--refuse` option if you are using an available unit other than `g`, `oz`, or `lb`. In the `starter_database` file, all available units (other than `g`, `oz`, and `lb`) already represent the edible portion of a food. For instance, the `large (3-1/4" dia)` available unit for Apples, raw, with skin is 223g. This means that a typical large apple weighs 223 grams, without the core, stem and seeds. If you use `--refuse` here Pantry will think the food weighs too little. Thus using the `--refuse` option typically makes sense only if you stick foods on a scale. It is quite handy if, for example, you weigh a chicken drumstick before you eat it. Otherwise you'd have to weigh the drumstick, eat it, and then weigh the bone to figure out how much the meat weighed.

The other approximations tool is the `--by-nut` option. If you use it then Pantry will automatically adjust the quantity of a food so that a nutrient you specify will be of an amount that you specify.

For example, let's say I have just eaten half a cup of Haagen Dazs chocolate ice cream. I look on the container and find that I've just eaten 270 calories. But the `starter_database` file does not have Haagen Dazs. It does have `Ice creams, chocolate, rich`, but it doesn't have the same number of calories per gram. That bothers me because I want to know exactly how many calories I ate.[6] One way I could deal with this discrepancy would be to create a custom food for Haagen Dazs chocolate ice cream. We'll learn how to do that later. But for now I decide that `Ice creams, chocolate, rich` is good enough for my purposes. I figure I will just record that I ate 270 calories of `Ice creams, chocolate, rich`:

[6] Actually `Ice creams, chocolate, rich` comes really close to Haagen Dazs, but since I'm so anal (wouldn't I have to be, to write a program like Pantry?) I want to be more exact.

Example 2.33. Using the `--by-nut` option

```
$ pantry --search name \
"Ice creams, chocolate, rich" \
--by-nut Calories 270 \
--print name --print info \
--goal Calories 2800 \
--print goals \
--change date 2008-06-20 \
--add diary master
Ice creams, chocolate, rich
105.882352941176478 g (105.882g)
Group: Sweets
Date: 2008-06-20
Nutrient name                              Amount %G    %T
-----------------------------------------------------------
Calories                                   270    10    100
```

As you can see Pantry adjusted the quantity of `Ice creams, chocolate, rich` so that it would be equal to 270 calories. `--by-nut` also lends itself to other uses, such as determining what quantity of a particular food will have a given amount of calories, fat, etc. so that you can meet whatever goals you have set for yourself.

The `paste` report

As you've been reading this chapter, one protest you might have is that a lot of Pantry commands require a lot of typing. I agree; the best solution, however, comes by using the features of your shell (such as functions, aliases, history, and variables) to help speed things up. We'll talk about those in the next chapter. Pantry does, however, have one innovation to help in a common situation. Often, you'll need to look through foods in the `master` collection in the `starter_database` file to find what you are looking for. Such a search involves two steps: first, you must find the right food; and second, often you must find the right available unit.

To merge these two steps into one, there is the `paste` report. It shows both the names of foods and their available units. The format of the report allows you to take the results and cut and paste them. Let's say for example I am looking for some sort of strawberry:

Example 2.34. The `paste` report

```
$ pantry --ignore-case \
--search name strawberries \
--search group fruit \
--print paste master | head
pantry -xsn "Strawberries, raw" -cu "NLEA serving" "master"
pantry -xsn "Strawberries, raw" -cu "cup, halves" "master"
pantry -xsn "Strawberries, raw" -cu "cup, pureed" "master"
pantry -xsn "Strawberries, raw" -cu "cup, sliced" "master"
pantry -xsn "Strawberries, raw" -cu "cup, whole" "master"
pantry -xsn "Strawberries, raw" -cu "extra large (1-5/8\" dia)" "master"
pantry -xsn "Strawberries, raw" -cu "g" "master"
pantry -xsn "Strawberries, raw" -cu "large (1-3/8\" dia)" "master"
pantry -xsn "Strawberries, raw" -cu "lb" "master"
pantry -xsn "Strawberries, raw" -cu "medium (1-1/4\" dia)" "master"
```

Then I can simply find the line with the right available unit and food name and paste that to my command line. I'll likely want to add additional options after I paste, such as `--change quantity`, `--add,` or `--print`. Unlike the units report, the paste report includes entries for the available units `g`, `oz`, and `lb`. Of course, to save space in the manual I stuck **`| head`** at the end; instead you'll likely want to send the output to your favorite pager, such as **less**.

In order to actually do the cutting and pasting, just use your favorite X terminal emulator, or the cutting and pasting features of GNU screen [http://http://www.gnu.org/software/screen/].

Saving and quitting

When you make changes to a database using the add option, those changes are made to the database that is currently in memory. Pantry does not write those changes to disk unless you tell it to. You can see whether you have unsaved changes to the currently open database by using the `--status` option:

Example 2.35. Seeing if there are unsaved changes

```
$ pantry --status
Current filename: /home/massysett/pantry-git/data/starter_database
Collections:
     8 diary (changed)
  7413 master
```

Two options, `--save` and `--save-as,` will save your Pantry database to a file.[7] `--save` will save the database to the filename shown in the Current filename: part of the status report; if there is no current filename, Pantry will complain with an error message. `--save-as` takes a single argument, which is the name of the file you wish to use. `--save-as` will save your database to that file, and it will also change the current filename so that the database will be saved to that same file if you later use the `--save` option.

Here, the current filename is `/usr/local/share/pantry/data/starter_database`. That's a location that is only writeable by the root user, so if I use the `--save` option, Pantry will give me an error and quit. So instead I need to use `--save-as` to save the database to a file in my home directory:

Example 2.36. Using the `--save-as` option

```
$ pantry --save-as ~/my_pantry_file
```

Finally you may want to shut down the server if you are done with Pantry. You do this with the `--quit` option: just run **`pantry --quit`**.

Additional commands to work with files

You've already learned all the options you'll typically need to work with Pantry files: `--open, --save,` and `--save-as`. I'll briefly mention a couple of others. First, one of Pantry's limitations is that it can work with only one database at a time, and one disk file holds one database. So there's no apparent way to take the contents of one database and merge it with another. Here's where the `--read` option comes in. The `--open` option reads the contents of a disk file into memory, clobbering whatever database was already in memory.[8] In contrast, the `--read` option takes the contents of a disk file and adds those contents to whatever is in memory. Using this you can, for example, merge the contents of two files.

[7] The database is written in plain ASCII text, but it's nothing you will want to edit directly.

[8] `--open` does not warn you before it clobbers what's in memory, even when the contents of the database in memory have not been saved. It assumes you know what you're doing.

The second option I'll mention here is simple: `--close` simply wipes the current database from memory, leaving you with a blank slate.

Oops! When things go wrong

Pantry takes a lot of command-line options, and sometimes you will get them wrong. Sometimes I get them wrong, and I wrote this thing. Sometimes you will also try to do something that Pantry does not allow. Examples include using a bad pattern for a regular expression (such as `ab(c`, which has an unmatched parenthesis) or giving a bad value for a numeric trait (such as attempting to change the `quantity` trait to `a1`). File operations have rich potential for errors--you might try to open a nonexistent file, open a file that is not a Pantry file, or save to a place where you don't have write permission.

Whenever Pantry can't do something for whatever reason, it will give you an error message and the client will quit with a non-zero exit status.[9] The server will keep running; it's only the client that will quit. Therefore you can easily try again.

If any error occurs, no changes will have been made to the database that is in memory or to any disk file.[10] That is, the `--add`, `--edit`, and `--delete` options will have had no effect. If an `--open`, `--save`, `--save-as`, or `--read` option fails for any reason, then the database that is in memory will remain unchanged.

The server should never quit unexpectedly. If it does, it is probably a bug; please report it to `<omari@smileystation.com>`.

[9] Currently the client will always exit with 1 if there was an error--any error. Maybe in future versions the exit code will change depending on the nature of the error.
[10] This is a dramatic improvement from earlier versions of Pantry, which had a variety of behaviors when errors happened.

Chapter 3. Pantry usage tips

There are a number of ways that you can make your use of Pantry more efficient, as this chapter will show you.

Use your shell

Pantry was designed to be used in concert with a good Unix shell. There are lots of good shells to pick from. GNU's Bourne Again Shell (**bash**) is the most common shell on Linux and it's probably the shell you are using if you do not know what shell you are using. **bash** is quite serviceable and powerful.

I'd like to recommend two alternatives to **bash**. **zsh** is an excellent shell for command-line junkies, and it is very well documented. **zsh** is available under a free software license, every major Linux distribution makes it available in its package repositories, and it is also available in Mac OS X. Another alternative is **fish**, the Friendly Interactive Shell. **fish** benefits from a sharp focus on usability. By breaking with compatibility with other Unix shells, **fish** is clear of clutter and predictable to use. **fish** is also free software. It's not as widely distributed as **zsh**, so you might have to compile it yourself.

If you are going to stick with **bash** or **zsh**, I highly recommend you read *From Bash to Z Shell: Conquering the Command Line*, by Kiddle, Peck and Stephenson. The authors are quite knowledgeable about Unix shells (Stephenson currently maintains **zsh**) and this book will teach you many useful tricks. It is well worth the price.

If you are a **zsh** user you're probably familiar with its powerful completion capabilities. You'll find a set of completions for **pantry** and **pantryd** at `$PREFIX/share/pantry/scripts/_pantry`.

History features

It will be particularly useful to you if you learn your shell's features for manipulating your command history. Often with Pantry I find myself re-running previous commands, with a minor tweak here and there. History features make this easy.

Aliases and functions

Another key shell feature is the ability to define aliases (which perform simple text substitution) and functions (which are programs in their own right.) I find both to be invaluable while using Pantry. I use functions to define my own custom commands to quickly add foods to a day's diary, something which I found to be nearly impossible with any other nutrient analysis software available. After using Pantry for a little while, you'll form your own ideas of what sort of functions will help you. For inspiration, you can take a look at the `$PREFIX/share/pantry/scripts/zsh_functions` file which is included in the Pantry distribution, where $PREFIX is the prefix you used when you ran the configure script. (By default it is `/usr/local`.) These are the functions I use. You might find them useful as-is, but I encourage you to modify them to fit your needs.

Variables and arrays

A final key shell feature is shell variables[1] and, in particular, arrays. The `--list` and `--goal` options are much more usable if you take advantage of arrays. For example, you might want to display several nutrients every time you print a goals report. As we saw in the last chapter, typing all these nutrients gets

[1] Or, what some shell documentation will call parameters.

tiresome very quickly. The solution is to put the nutrients you want into an array.[2] Then, instead of typing all of them at the command line, you simply type the name of your array instead.

For reasons related to word splitting [http://zsh.dotsrc.org/FAQ/zshfaq03.html#l18], I recommend you use **zsh** or **fish** if you're going to become an array user, as their syntax is a bit friendlier than that of **bash** and **ksh**.

Short options and abbreviations

For instructional purposes, all the examples in this manual use double-dashed long options. However, Pantry has single-dashed short options for all options you'll commonly use. To see what these are, refer to the Pantry man page or to **`pantry --help`**.

In many places, Pantry will accept abbreviations instead of full words. For example, for long options, you may type just enough to make yourself unambiguous. This means you can type pantry `--q` instead of pantry `--quit`. (pantry `--qu` or pantry `--qui` will work too.) You can also do this when you must specify a trait name. Because the first letter of each trait is unique, you can specify just a single letter. Thus, you can type **`pantry --search n bananas master`** instead of **`pantry --search name bananas master`**. Furthermore, you can abbreviate the names of reports for the `--print` option, and you can abbreviate collection names if you are specifying which collections to search (you can't abbreviate the argument to the `--add` option.)

That gives you lots of abbreviating options. For instance, instead of typing **`pantry --ignore-case --search name bananas --search group fruits --key name --print name master`**, you can instead run **`pantry -isn bananas -sg fruits -kn -pna m`**.[3]

Files and collections

Pantry is easiest to work with if you keep all your data in a single Pantry file. That's because Pantry does not make it very easy for you to move data from one file to another. Pantry has collections so that you can sort foods within a file in a way that you see fit. I find it easiest to keep three collections in my Pantry file. In two of these collections, `master` and `quick`, every food has a unique name trait. That makes it easy for me to find foods using simply their `name` trait. There are over 7000 foods in my `master` collection; it has every food that's in the `starter_database` file as well as other foods that I created on my own (we'll talk later about how you can create foods and recipes.) A second collection in my file, `quick`, holds only foods that I use frequently. There are about fifty foods in here currently. As with the `master` collection, every food in my `quick` collection has a unique `name` trait. Many of the foods in my `quick` collection already have their `quantity` and `unit` traits set to those that I commonly eat. Finally, I have a `diary` collection. When I add foods to my diary collection I set the date and meal traits to appropriate values.

This is just the way I do things; you might find a better way. For instance, maybe you will start a different collection for every day, or for every week. You might not find it necessary to keep a separate `quick` collection. You might keep a diary in the same collection as you keep all your other foods. It's up to you.

Using traits

For most traits, Pantry places no constraints on what values you can use. This gives you a great deal of latitude. For example, if you have several foods you eat together, one way to easily account for them in a diary is if you create a recipe, which we will talk about later. Another way is to simply change the `group`

[2] Under certain circumstances you can do this using regular, "scalar" variables rather than arrays, but you'll be making life hard on yourself.

[3] Pantry is liberal about whether spaces or equal signs must separate option arguments from the option itself; for instance, you can type either **`-pname, -p name`**, **`--print name`**, or **`--print=name`**.

traits of these foods so they are unique and identical. For example, I often eat a particular cereal with a particular milk, so in my `quick` collection I have changed the group traits of these foods to `Cereal and milk`.

You can use whatever values you want for the date trait, but you will find that it is easiest to sort dates if you use a YYYY-MM-DD format.

Chapter 4. Creating new foods, and changing the nutrients and available units of existing foods

So far we've only dealt with foods that already exist in the `starter_database` file, which comes from data derived from the USDA National Nutrient Database for Standard Reference. We've copied these foods from one collection to another, often changing their traits along the way. But we haven't changed the nutrient content of these foods.

But what if you want to track information about a food that is not already in the USDA database? You might find there is already a food in the database that is close enough for your purposes. However, with Pantry you can easily create your own foods with their own nutrient data. Such foods are indistinguishable from those in the `starter_database` file--except for the fact that you created them yourself. In this chapter you'll learn how to create foods of your own.

You'll want to create a food if you already have nutrition information for what you are creating--from a label, say, or from a website. If you are combining multiple ingredients, you'll want to use a recipe instead, which we will discuss in the next chapter.

Create the food and its traits

You'll create foods with Pantry the same way you do anything else with Pantry: from the command line.[1] In this chapter we'll enter nutrition information for Heritage Flakes. There is something close in the `starter_database` file--`Cereals ready-to-eat, wheat and malt barley flakes`--but Heritage Flakes is a little higher in some nutrients, such as protein, and has twice as much fiber. You can get the nutrition information here [http://www.naturespath.com/products/cold_cereals].

To create a new food, use the `--create` option. This will give you a buffer with a single, blank food. Then you can use the `--change` option to change the traits to something appropriate, and the `--add` option to add the food to a collection. You can use the `--print` option to print the buffer, as usual. That gives us:

Example 4.1. Creating a new food

```
$ pantry --create \
--change name "Heritage Flakes" \
--change quantity 30 \
--change unit '^g$' \
--print name --print info \
--add master
Heritage Flakes
30 g (30g)
```

We changed the quantity to 30 grams because that is the quantity that the food label is based upon.[2] It's essential that the quantity and units be correct when you add nutrient information.

[1] This differs dramatically from older versions of Pantry, which used XML files for this purpose.

[2] You might wonder why we don't say that the quantity is `3/4` and the unit is `cup`. That's because later we'll discuss adding available units, which is what you'll have to do so that Pantry knows how much a cup of Heritage Flakes weighs.

Add available units

Next you can add available units to your food, if you wish. Pantry will automatically supply your food with the available units `g`, `oz`, and `lb`, so there is no need to add those. From the label we see that 3/4 of a cup of Heritage Flakes weighs 30 grams. This means a cup weighs 40 grams. Using this knowledge we can use the `--change-avail-unit` option. Its first argument is the name of an available unit that you wish to create or change, and its second argument is the weight of that available unit, in grams.

Example 4.2. Adding an available unit

```
$ pantry --exact-match \
--search name "Heritage Flakes" \
--change-avail-unit cup 40 \
--edit master
```

Create nutrients

Next you can add nutrients. You can use whatever nutrient names you wish, but for consistency's sake you might wish to use names that are identical to those used in the `starter_database` file. (To get a list of all nutrient names used in the `master` collection of the `starter_database` file, you can run **`pantry --print Nuts master`**. That command might take several seconds to run, especially on a slow machine.)

On U.S. food labels, many nutrients are given as percentages. You might just enter these percentages if you wish. You can convert them to actual values using this chart [http://www.fda.gov/FDAC/special/foodlabel/dvs.html]. Right now we'll just skirt that issue by entering only macronutrients.[3] Be sure to use the `--edit` option so your changes are permanent.

Example 4.3. Adding nutrients

```
$ pantry --exact-match \
--search name "Heritage Flakes" \
--change-nut "Calories" 120 \
--change-nut "Total Fat, g" 1 \
--change-nut "Saturated Fat, g" 0 \
--change-nut "Trans Fat, g" 0 \
--change-nut "Cholesterol, mg" 0 \
--change-nut "Sodium, mg" 130 \
--change-nut "Total Carbohydrate, g" 24 \
--change-nut "Dietary Fiber, g" 6 \
--change-nut "Sugars, g" 4 \
--change-nut "Protein, g" 4 \
--edit master
```

Check your work

That should do it. Let's make sure:

[3] Pantry comes with a script that helps you enter new foods, and it converts these percentages to actual values. We'll talk about this script later.

Example 4.4. Seeing the new food

```
$ pantry --exact-match \
--search name "Heritage Flakes" \
--print name --print info --print nuts \
master
Heritage Flakes
30 g (30g)
Nutrient name                                Amount %G    %T
-------------------------------------------------------------
Calories                                     120          100
Cholesterol, mg                              0
Dietary Fiber, g                             6            100
Protein, g                                   4            100
Saturated Fat, g                             0
Sodium, mg                                   130          100
Sugars, g                                    4            100
Total Carbohydrate, g                        24           100
Total Fat, g                                 1            100
Trans Fat, g                                 0
```

In this example, for instructional purposes we used more than one pantry command to add the traits, nutrients, and available units. You can do all this in one command, however. You can even add the food to collections and `--print` it, all in the same command.

The addFoods script

You might find it more convenient to enter the commands required into a simple shell script, since there are so many to type. Alternatively, you can use the **addFoods** script, which you will find in `$PREFIX/share/pantry/scripts/addFoods`, where $PREFIX is the prefix you used when you installed Pantry. (By default this is `/usr/local`.) This script takes the name of an input file, which is simply a number of **zsh** variables containing the data for the food. This script also automatically converts micronutrients, which are expressed on US food labels as percentages, to actual values. To get an idea how the script works, just take a look at the script itself. Alternatively, you will find a number of sample input files for this script in the `$PREFIX/share/pantry/examples/foods` directory. To get started you can just model your input after one of these files.

The **addFoods** script is written for **zsh**, so you'll have to install **zsh** in order to use it.

Commands to edit the nutrients and available units of foods

Pantry has a number of options to edit the nutrients and available units of existing foods. Changing nutrients and available units is similar to changing traits: Pantry makes the changes you wish, but only to foods that are in the buffer. To make your changes permanent, you need to use the `--add` or `--edit` option. We've already seen the `--change-nut` and `--change-avail-unit` options. These work not only for adding new nutrients or available units, but also for modifying existing ones. Remember that the first argument to `--change-nut` and `--change-avail-unit` is not a regular expression--it is the exact name of the nutrient you wish to add or change, and it is always case sensitive, even if you use the `--ignore-case` option.

If you wish to change the name of an existing nutrient, use `--rename-nut.` It takes two arguments. The first is a pattern for the nutrient name you wish to change. As with the `--search` option, this is a

regular expression, unless you use the `--exact-match` option. `--rename-nut` also respects the `--ignore-case` option. The second argument is the new nutrient name.

Example 4.5. Using `--rename-nut`

```
$ pantry --search name "Heritage Flakes" \
--rename-nut "Total Carbohydrate, g" Carbs \
--print name --print info --print nuts \
master
Heritage Flakes
30 g (30g)
Nutrient name                              Amount %G    %T
-----------------------------------------------------------
Calories                                   120          100
Carbs                                      24           100
Cholesterol, mg                            0
Dietary Fiber, g                           6            100
Protein, g                                 4            100
Saturated Fat, g                           0
Sodium, mg                                 130          100
Sugars, g                                  4            100
Total Fat, g                               1            100
Trans Fat, g                               0
```

For each food in the buffer, the pattern given in the first argument to `--rename-nut` must match either no nutrients or one nutrient of each food. If the pattern matches no nutrients, Pantry will make no change to that food. If the pattern matches one nutrient, then Pantry will rename that nutrient. If the pattern matches more than one nutrient, Pantry will give you an error message and quit.

The `--rename-avail-unit` option renames available units (except for `g`, `oz`, and `lb`, which you cannot rename); otherwise, it is identical to the `--rename-nut` option. As with the `--rename-nut` option, the `--rename-avail-unit` pattern must match either zero or one available units for every food in the buffer (excluding `g`, `oz`, or `lb`); otherwise, Pantry will give you an error and quit.

Example 4.6. Using `--rename-avail-unit`

```
$ pantry --search name "Apples, raw, with skin" \
--rename-avail-unit large big \
--print name --print units master
Apples, raw, with skin
   NLEA serving (242g)
   big (223g)
   cup slices (109g)
   cup, quartered or chopped (125g)
   extra small (2-1/2" dia) (101g)
   medium (3" dia) (182g)
   small (2-3/4" dia) (149g)
```

To delete nutrients, use `--delete-nuts`. This option takes one argument, a pattern for nutrients you wish to delete. This pattern is a regular expression, unless you use `--exact-match`, and it respects `--ignore-case`. It deletes every nutrient that matches the pattern, and it doesn't complain if no nutrients match the pattern:

Example 4.7. Using `--delete-nuts`

```
$ pantry --search name "Heritage Flakes" \
--delete-nuts Fat \
--print name --print nuts \
master
Heritage Flakes
Nutrient name                              Amount %G    %T
-----------------------------------------------------------
Calories                                   120          100
Cholesterol, mg                            0
Dietary Fiber, g                           6            100
Protein, g                                 4            100
Sodium, mg                                 130          100
Sugars, g                                  4            100
Total Carbohydrate, g                      24           100
```

The `--delete-avail-units` option deletes available units, other than `g`, `oz`, and `lb`. In all other respects it is identical to the `--delete-nuts` option:

Example 4.8. Using `--delete-avail-units`

```
$ pantry --search name "Apples, raw, with skin" \
--delete-avail-units large \
--print name --print units master
Apples, raw, with skin
   NLEA serving (242g)
   cup slices (109g)
   cup, quartered or chopped (125g)
   extra small (2-1/2" dia) (101g)
   medium (3" dia) (182g)
   small (2-3/4" dia) (149g)
```

Chapter 5. Creating recipes

Creating new foods works great if you already know the nutrient content of a food--if, for example, you can pull the information from a label. Recipes, on the other hand, are ideal if you are creating a food from other foods and you don't know the nutrient content of the final food.

In Pantry, recipes are mostly like other foods. They are different in that they have two additional traits: `yield` and `instructions`. `yield` indicates the total weight, in grams, of one batch of this recipe. This is quite handy because recipes often lose water as you cook them. Of course you can only know the exact yield after you have cooked a recipe and have weighed it. If you do not supply a `yield` trait for a recipe, Pantry will guess the yield by adding up the weight of all the recipe's ingredients. `instructions` is simply food preparation notes, or whatever you want.

Create the recipe and its traits

This step is similar to the first step of creating a food: you use the `--create` option, and then you set the traits as you wish.

Example 5.1. Creating a recipe

```
$ pantry --create --change name "Easy Corn Bread" \
--change group "Baked Products" \
--change quantity 50 \
--change unit '^g$' \
--change instructions 'Heat oven to 400
degrees F. Grease 8- or 9-inch pan. Combine
dry ingredients. Stir in milk, oil, and
egg, mixing just until dry ingredients are
moistened. Pour batter into prepared pan.
Bake 20 to 25 minutes or until light golden
brown and wooden pick inserted in center
comes out clean. Serve warm.' \
--add master
```

Add ingredients

To add ingredients to a food, you use the `--add-ingredients` option, which takes a single argument, which is the name of the collection to add foods from.[1] Pantry adds all the foods from the collection as ingredients. Thus, the easiest way to create recipes is to add all the ingredients to a temporary collection--`ingredients`, perhaps--and then use that collection as an argument to `--add-ingredients`. You can just delete the collection with `--delete` when you are done.

[1] You can abbreviate the argument to the shortest unambiguous name for the collection, just as you can abbreviate collections that are positional arguments to the pantry command.

Example 5.2. Adding ingredients to a single collection

```
$ pantry --exact-match \
--search name \
"Wheat flour, white, all-purpose, enriched, unbleached" \
--change quantity "1 1/4" --change unit cup \
--add ingredients master

$ pantry --exact-match \
--search name \
"Cornmeal, whole-grain, yellow" \
--change quantity "3/4" --change unit cup \
--add ingredients master

$ pantry --exact-match \
--search name \
"Sugars, granulated" \
--change quantity "1/4" --change unit cup \
--add ingredients master

$ pantry --exact-match \
--search name \
"Leavening agents, baking powder, double-acting, straight phosphate" \
--change quantity "2" --change unit tsp \
--add ingredients master

$ pantry --exact-match \
--search name \
"Salt, table" \
--change quantity "1/2" --change unit tsp \
--add ingredients master

$ pantry --exact-match \
--search name \
"Milk, reduced fat, fluid, 2% milkfat, with added vitamin A" \
--change quantity 1 --change unit cup \
--add ingredients master

$ pantry --exact-match \
--search name \
"Oil, corn and canola" \
--change quantity "1/4" --change unit cup \
--add ingredients master

$ pantry --exact-match \
--search name \
"Egg, whole, raw, fresh" \
--change quantity 1 --change unit large \
--add ingredients master
```

Okay, now we can add the ingredients to the food itself. After adding them to the food we have no use for the collection so we can just delete it.

Example 5.3. Add ingredients to the food

```
$ pantry --search name "Easy Corn Bread" \
--add-ingredients ingredients \
--edit master

$ pantry --delete ingredients
```

Add available units

You do this just as you would for a food. We won't add any available units here, but you will often find it is handy to add an available unit for serving. To do this you can simply divide a recipe's yield by the number of servings. In the next section you'll see how to find a food's estimated yield if you haven't weighed the yield yourself.

Seeing your new recipe

In order to see your new recipe, you use the `recipe` report. This report shows the `yield` trait, ingredients, and `instructions` trait for a recipe. Recipes and foods are fully interchangeable in Pantry--for instance, a recipe can be the ingredient for another recipe.

Example 5.4. The `recipe` report

```
$ pantry --search name "Easy Corn Bread" \
--print name --print info --print recipe \
master
Easy Corn Bread
50 g (50g)
Group: Baked Products
Yield: 659.95g (estimated)
1 1/4 cup (156g, 5.51oz) Wheat flour, white, all-purpose, enriched, unbleached
3/4 cup (92g, 3.23oz) Cornmeal, whole-grain, yellow
1/4 cup (50g, 1.76oz) Sugars, granulated
2 tsp (9g, 0.32oz) Leavening agents, baking powder, double-acting, straight phosph
1/2 tsp (3g, 0.11oz) Salt, table
1 cup (244g, 8.61oz) Milk, reduced fat, fluid, 2% milkfat, with added vitamin A
1/4 cup (56g, 1.98oz) Oil, corn and canola
1 large (50g, 1.76oz) Egg, whole, raw, fresh

Heat oven to 400
degrees F. Grease 8- or 9-inch pan. Combine
dry ingredients. Stir in milk, oil, and
egg, mixing just until dry ingredients are
moistened. Pour batter into prepared pan.
Bake 20 to 25 minutes or until light golden
brown and wooden pick inserted in center
comes out clean. Serve warm.
```

For ingredients, Pantry prints the `quantity` trait, the `unit` trait, and the `name` trait. Pantry will print the `comment` trait's value in parentheses, if the comment trait is set at all. Pantry prints ingredients by sorting their `order` traits in ascending order, even though Pantry does not show the order traits for the ingredients. So, if you want to ensure that your ingredients are shown in a certain order, set your `order` trait for each ingredient accordingly.

A recipe's nutrient makeup is determined by all its ingredients. The nutrient makeup is scaled depending upon the `yield` trait, which is why Pantry will be more accurate if you weigh the yield of a recipe and enter it in. If the yield trait of a recipe is not set (of if you explicitly set it to zero) then in the `recipe` report Pantry will show an estimate for the recipe's yield, as it has done here. Pantry gets this estimate by adding the weight of all of a recipe's ingredients. Typically this estimate will be high, as cooking typically causes some of the water in a recipe to evaporate.

The `addRecipes` script

Since you have to type so much in order to add a recipe, you might want to put the commands in a simple shell script. Also, just as there is an `addFoods` script, there is also an `addRecipes` script. The script allows you to define a few shell variables in a file, which you then feed to the script in order to create your recipe. You'll find it at `$PREFIX/share/pantry/scripts/addRecipes`, and there are sample input files at `$PREFIX/share/pantry/examples/recipes`, where $PREFIX is the prefix you used when you installed Pantry. (It defaults to `/usr/local`.)

Like the **addFoods** script, the **addRecipes** script is written for **zsh**.

If you use the `addRecipes` script, also take a look at the `ingrlist` script, which will help you as you look up ingredients to add to input files for `addRecipes`.

Editing recipes

Recipes and foods in Pantry are nearly fully interchangeable. You can edit the traits and available units for a recipe in the same way as you do for a food. Nutrients, however, are a bit different. In a regular food, you input all the nutrient data. For a recipe, the nutrient data is typically calcualted from the ingredients. You can use `--change-nut` on a recipe. This will create a nutrient that will override the automatically-calculated nutrient of the same name, if there is one. For recipes, `--rename-nut` only works for nutrients you have entered using `--change-nut,` and `--delete-nuts` only deletes nutrients that you have set using `--change-nut` or `--rename-nut.` If you delete a nutrient you have set using `--change-nut` or `--rename-nut`, Pantry will then revert to using the automatically-calculated nutrient, if there is one.

Appendix A. Pantry limitations

Pantry isn't perfect. It has some limitations you might want to know about.

No multibyte or locale awareness

Pantry is horribly U.S. centric. It knows nothing about multibyte character encodings, such as UTF-8. Furthermore, it knows nothing about locales, so collation [http://en.wikipedia.org/wiki/Collation] is performed in what has sometimes been called ASCIIbetical order.

Strictly speaking you could use any encoding with Pantry, as long as it is a single-byte one. For instance, Pantry would have no problem with characters from the ISO-8859-1 character set. However, I recommend you stick with US-ASCII, for two reasons. First, most Linux distributions these days use UTF-8 for their terminal encoding. If you use characters solely from the ASCII character set [http://en.wikipedia.org/wiki/ASCII], then UTF-8 is identical to ASCII. However, if you use characters from ISO-8859-1 [http://en.wikipedia.org/wiki/ISO/IEC_8859-1], and your terminal is set to UTF-8, then you will really be using multibyte characters, which Pantry does not know how to handle. Second, even if you get your terminal set up correctly to use a non-ASCII single-byte encoding, your file may behave differently if you later use it on a different computer with a different terminal encoding. ASCII, on the other hand, is an encoding which will work nearly anywhere.

Converting Pantry to use multibyte encodings correctly would not be terribly difficult. Pantry could ensure that all input data is converted to UTF-8 for internal use, and then convert it to the proper encoding when sent to the user's terminal. Another issue would be regular expressions. I've never seen regular expression routines from a UNIX standard C library that claim to be multibyte aware, so they probably are not. A straightforward solution would be to use PCRE, which can handle UTF-8.

The reason I haven't made Pantry handle multibyte properly is because all the documentation is in English and the interface is in English. This is not likely to change soon, as Pantry has no community of users to translate it, and English is the only language I know. If you are fluent in written Enlish, then likely it is not much of a limitation to use only US-ASCII with Pantry. I could one day make Pantry multibyte aware, if it seemed someone cared about that and if I had lots of free time.

High RAM usage

Pantry can use lots of memory. Well, whether it is "lots" depends on what you think a lot is. When I open the `starter_database` file on my computer, Pantry consumes about 30 MB of memory. I don't think that is too bad, but then, my least capable computer has 512 MB of RAM, and typically I don't even use half of that.

Pantry consumes so much memory because it stores so much data. Most of the foods in the `starter_database` file have a few dozen nutrients; that's what consumes the vast majority of storage space. Therefore, if you want to keep all the foods in the `starter_database` file but you want to cut memory usage, the easiest way to do it is to delete nutrients you don't need, which of course is easy with the `--delete-nuts` option. The less data you store, the smaller Pantry's memory footprint will be.

A great deal of code in Pantry 26 was rewritten so that Pantry is more frugal with memory. Though the starter_database file currently consumes 30 MB of memory on my machine, with older versions of Pantry this was 55 MB. Pantry 26 is also a bit faster than older versions of Pantry, too.

Only one file at a time may be open

In Pantry each server can work only with one file at a time. This could possibly be changed, but right now I don't see how changing this would do anything other than make the user interface more complex.

Size of collections, number of collections, number of nutrients, etc.

There are limitations to how many foods you can have in a collection, how many collections you may have in a single database, etc. These limitations will vary depending upon the nature of your standard C++ library. On my machine, for example, I can have (in theory) about 4 billion collections. Each collection can have a much lower number foods: only about 1 billion. For all practical purposes you'd run out of memory before you hit these limitations.

Numbers are approximations

Gasp! This is a nutrition analysis software package, and the numbers are approximations?! Yes. Internally Pantry uses floating point numbers. Such numbers are, by their very nature, approximate [http://en.wikipedia.org/wiki/Floating_point]. We're talking typical approximations on the order of, for example, fractions of a calorie--not tens of calories or even a single calorie.

Though Pantry could be made more precise, this would come at the expense of more memory usage and more code. Furthermore, the only result would be precise calculations of nutrition data--data which is, by its very nature, also approximate. So at this point I'm inclined to say that Pantry's approximations are "good enough."

Appendix B. How the USDA National Nutrient Database for Standard Reference became the `starter_database` file

If you are curious about how the data from USDA's National Nutrient Database is translated into a form Pantry can understand, and whether any data was lost along the way, you've come to the right place.

USDA releases its database in the form of about a dozen plain text files, which you can download from the USDA website [http://www.ars.usda.gov/main/site_main.htm?modecode=12354500]. If the data in Pantry looks familiar, it's because this USDA data forms the basis for pretty much any food database you will find in the US, either on the Web or on your desktop, in free software and in proprietary products.

I have written a **zsh** script to convert the USDA database into a series of over 7,000 Pantry commands--one command for each food in the database. You can find the script at `$PREFIX/share/pantry/scripts/convert_sr.zsh`.[1] This script renames some nutrients from names like `Fatty acids, total trans` to `Trans Fat`. It also discards some of the gram weights (or, in Pantry lingo, available units) from the `WEIGHT.txt` file. For example, the `WEIGHT.txt` file contains weights such as `lb` which Pantry provides automatically. Also, it provides some weights where the quantity is more than one--it might have a weight for, say, 2 slices of pizza rather than for 1 slice of pizza. I thought hard about what to do with data like this before finally deciding to just discard it. The only accurate way to keep this data would be to manually go through it and edit the descriptions accordingly.

The script spits the Pantry commands to standard output, which I then save to a file and use to create the `starter_database` file. You can even do it yourself. You might do this if, for instance, USDA has released a new database and you don't want to wait for me to get around to releasing a new version of Pantry. Beware, though: the **convert_sr.zsh** file works fine with SR20 and SR21 (USDA calls the database simply "SR"), but future SR releases might change their format and break the script. I doubt this would happen, but it's possible. Also, an earlier release of SR20 had small irregularities in the data that completely broke this script, but a subsequent release of SR20 fixed this problem.

If you want to see a list of all the nutrients in the `starter_database` file, open it and run **`pantry -p Nuts master`**. It will be obvious how these nutrients match up with those in SR, even though I changed the names a bit. This command may take a few seconds to run, especially on slow machines.

[1] This is a huge change from earlier versions of Pantry, which included Python code to import the USDA database. I've found these sorts of tasks to be easiest with **zsh**, though ksh would work fine too. Maybe Perl would work well, but I've never had any desire to learn Perl. C++ would be very nasty for a task such as this.

Appendix C. Reference pages

This appendix contains reference pages for the commands that Pantry uses. They are identical to the man pages; indeed, the man pages and this document are generated from the same source.

Name

pantry — nutrient tracking and analysis

Synopsis

`pantry` [options...] [*`collection`*...]

Description

pantry copies foods from *`collection`*s into a buffer. All foods are copied, unless one or more `--search` options are specified, in which case only foods that match the patterns of every `--search` option are copied. **pantry** then changes every food in the buffer using any `--change` options specified as well as any options for changing nutrients, available units, and ingredients.

If `--edit` or `--delete` is specified, **pantry** deletes the unchanged foods from the corresponding original *`collection`*s. If `--edit` is specified, **pantry** adds changed foods to corresponding original *`collection`*s.

If `--print` *`REPORT`* is specified, buffer is printed using *`REPORT`* Buffer is unsorted unless one or more `--key` *`TRAIT`* is specified. Sorting may be affected by one or more `--list` options.

If `--add` *`COLLECTION`* is specified, each food in the buffer is added to *`COLLECTION`*.

pantry communicates with a **pantryd** server, see pantryd(1).

Options

`TRAIT` is any one of:

- name
- group
- quantity
- unit
- date
- meal
- comment
- order
- refuse
- percent-refuse
- yield
- instructions

`TRAIT` names may be abbreviated.

PATTERN is an extended regular expression, unless `--ignore-case` is on.

Most options other than file and server control options may appear on the command line more than once; the effect of multiple options is cumulative.

File and server control

`--open` *filename*	Close currently open database and open *filename*
`--read` *filename*	Append contents of *filename* to currently open database
`--close`	Close currently open database
`--save`	Save currently open database
`--save-as` *filename*	Save currently open database under a different *filename*
`--quit`	Shut down server
`--status`	Display server status and database information: names of collections in database and the number of foods in each collection, and what filename will be used if `--save` option is used

Food selection

`--search` *TRAIT PATTERN*, `-s` *TRAIT PATTERN*	Include in buffer only foods whose *TRAIT* matches *PATTERN*.

Changing food traits

`--change` *TRAIT string*, `-c` *TRAIT string*	Change *TRAIT string*. If *TRAIT* is *unit*, *string* is a pattern that must match exactly one of the food's available units. If *TRAIT* is *quantity* or *unit*, *string* must be convertible to a non-negative number. If *TRAIT* is *percent-refuse*, *string* must be convertible to a number between 0 and 100, inclusive.
`--refuse`, `-r`	Reduce food quantity by percent refuse.

Changing nutrients, available units, and ingredients

`--change-nut` *string number*	Change nutrient whose name is *string* to *number*.
`--rename-nut` *pattern string*	Rename nutrient whose name matches *pattern* to *string*.
`--delete-nuts` *PATTERN*	Delete all nutrients matching *PATTERN*.
`--change-avail-unit` *string number*	Change available unit whose name is *string* to *number*.
`--rename-avail-unit` *pattern string*	Rename available unit whose name matches *pattern* to *string*.
`--delete-avail-units` *PATTERN*	Delete all available units matching *PATTERN*.

`--add-ingredients` *`COLLECTION`*
: Add ingredients from *`COLLECTION`*.

`--delete-ingredients` *`PATTERN`*
: Delete ingredients whose name trait matches *`PATTERN`*.

Search and edit options

`--ignore-case`, `-i`
: All patterns are case insensitive.

`--exact-match`, `-x`
: All patterns must exactly match their subjects (turns off regular expressions).

`--edit`
: Edit foods in place

`--delete`
: Delete matching foods

`--create`
: Create a new food from scratch. When this option is used, Pantry does not copy foods from any collections specified on the command line, and the `--edit` and `--delete` options are ignored.

`--limit` *`number`*
: Limit number of foods in buffer to *`number`*

Reporting

`--print` *`report`*, `-p` *`report`*
: Print *`report`* (see "Reports" section below)

`--key` *`TRAIT`*, `-k` *`TRAIT`*
: Use *`TRAIT`* as a sorting key. If the first letter of *`TRAIT`* is lowercase, sort in ascending order; if the first letter of *`TRAIT`* is uppercase, sort in descending order.

`--goal` *`nutrient-name amount`*, `-g` *`nutrient-name amount`*
: Add a nutrient intake goal for use by the nutrient-related reports, where *`nutrient-name`* is the nutrient for which you wish to add a goal, and *`amount`* is a string, convertible to a non-negative number, that is the amount of the goal.

`--list` *`TRAIT string`*, `-l` *`TRAIT string`*
: Add *`string`* to the list of strings that will be used when *`TRAIT`* is used as a sorting key. When *`TRAIT`* is equal to one of these values, it will be sorted in the order specified.

Adding results to collections

`--add` *`COLLECTION`*, `-a` *`COLLECTION`*
: Add buffer to *`COLLECTION`*.

`--auto-order`, `-o`
: When adding each food to collections specified with `--add`, **pantry** will search the collection for other foods with identical `date` and `meal` traits. The result will be sorted in ascending order by the *`order`* trait. If the highest food's *`order`* trait matches the regular expression `^[0-9]4$`, then **pantry** will take the highest food's *`order`* trait, remove any leading zeroes, remove the last digit, and increment the result by one. The result is multiplied by ten, and then is left-padded with zeroes so that it is four characters long. **pantry** will then change the `order` trait of the food to the result before adding it to the collection.

If there are no foods with identical `date` and `meal` traits, then **pantry** will set the food's `order` trait to `0010`.

Meta

`--help`, `-h` Display brief help.

`--version`, `-v` Display Pantry version.

`--copyright` Display copyright information.

Reports

Two types of reports are available. Food reports are printed once per food in the buffer. Summary reports are printed once for the entire buffer. To print more than one report, use multiple `--print` options. Report names may be abbreviated with an unambiguous specification of the first letters of the report. The following reports are available:

Food reports

`name` Food names

`info` All traits other than the *`name`*, *`yield`*, and *`instructions`*.

`recipe` *`yield`* and *`instructions`* traits, and the recipe's ingredients. Ingredients are sorted according to their *`order`* traits.

`units` Available units. *`g`*, *`oz`*, and *`lb`* are not printed as these are available for every food.

`paste` Each food name, printed with one available unit per line; quoted so that output may be easily pasted into subsequent **pantry** commands.

`blank` A blank line

`nuts` All of a food's nutrients

`goals` Nutrients for which there is a goal specified with `--goal`

Summary reports

`groups` Number of foods in each food group, as specified by each food's *`group`* trait

`Nuts` Sum of all nutrients in the buffer

`Goals` Sum of all nutrients in the buffer for which there is a `--goal`

Environment variables

PANTRY_SOCKET If this environment variable is specified, **pantry** will use the path contained therein as the filename of the socket where it will attempt to connect to the **pantryd** server. PANTRY_SOCKET should contain the entire path (beginning with `/`) and filename, not just the directory for the socket. The server must have already been started with an identical value in its PANTRY_SOCKET environment variable.

If this environment variable is not specified, **pantry** will attempt to connect to a server at the socket named `$HOME/.pantrySocket`.

Bugs

Please help find them. Report bugs to `<omari@smileystation.com>`.

Pantry has known limitations; see the Pantry User Guide for details.

Pantry home page

http://www.smileystation.com/pantry

Name

pantryd — nutrient tracking and analysis server

Synopsis

`pantryd` [options...]

Description

pantryd is the server with which **pantry** communicates. **pantryd** must be running in order for any **pantry** commands to work.

To start a server, simply execute **`pantryd`**. This will start a new server in the background, returning you to your command prompt. This server will keep running even if you logout of the shell from which you started the server. To kill the server, run **`pantry --quit`**.

Options

`--foreground`, `-f`	Run server in the foreground (useful for debugging)
`--help`, `-h`	Show brief help and exit

Environment variables

PANTRY_SOCKET If this environment variable is specified, **pantryd** will use the path contained therein as the filename of the socket where it will listen for connections. PANTRY_SOCKET should contain the entire path (beginning with `/`) and filename, not just the directory for the socket. To communicate with the server, you'll need to make sure that subsequent invocations of **pantry** also have their PANTRY_SOCKET environment variable set to an identical value.

If this environment variable is not specified, **pantryd** will listen for connections on a socket named `$HOME/.pantrySocket`.

Bugs

Report any bugs to <omari@smileystation.com>.

Pantry home page

http://www.smileystation.com/pantry