Pantry User Guide

Omari Norman

Pantry User Guide

by Omari Norman

Table of Contents

List of Examples

About Pantry

As you can see, this user manual is of some length. Right now you are probably wondering what Pantry does, whether it is right for you, and how it compares to similar programs.

1. What Pantry is

Pantry is a command-line oriented nutrient analysis program. That's right, command-line oriented. What's more, Pantry is a true command-line program: there are no menus, there are no prompts. Instead, you simply type commands from your shell prompt, and Pantry does what you ask it to do, displaying results if you have asked it to do that. Thus, Pantry is different from other programs which run in a text-based terminal but which also present the user with menus; Wikipedia calls such programs text-user interface (http://en.wikipedia.org/wiki/Text_user_interface) programs.

In addition to using Pantry from your shell prompt, you also interact with it through XML (http://en.wikipedia.org/wiki/Xml) files. Using XML, you can edit Pantry's configuration file. You can also add nutrient information for custom foods (though Pantry includes nutrient information for over 7,000 foods to get you started) and recipes using XML.

2. 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.

3. Pantry 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.

A final disadvantage of using Pantry is that it is still new. I am still tweaking it, making changes, adding features, and improving the documentation. But perhaps this is not such a disadvantage: software that improves is nice. If you have any features that you would like, ask!

4. 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://www.lafn.org/~av832/).

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).

5. Getting support and reporting bugs

For help using Pantry, join the Pantry users email list. Signup info is here (https://lists.sourceforge.net/lists/listinfo/pantry-users). You can also report bugs by sending an email to <pantry-users@lists.sourceforge.net>. You don't have to subscribe to the list to send emails to it (but of course you have to subscribe to get replies!)

Chapter 1. Installing Pantry

This section will tell you how to install Pantry on Linux. Pantry is written in Python (http://www.python.org), a cross-platform language. Therefore, it should run on any Unix-like operating system, including any of the BSDs and Mac OS X. However, I don't know anything about the BSDs or OS X, so I can't give you instructions for those. However, Pantry is distributed using the Python distutils (http://docs.python.org/inst/inst.html), which is a standard way to distribute Python software, so if you familiarize yourself with distutils you should be able to install Pantry on any Unix-like operating system.[1]

I'll assume you're comfortable using command-line tools--if you weren't, you probably wouldn't be attempting to use a command-line nutrient analysis program. The following sections will tell you step by step how to install Pantry.

1.1. Installation instructions

Warning for experienced users: Pantry (and most Python programs) do not use the GNU configure and build system--that is, **./configure; make; make install** will *not* work, so read the directions below.

1. Unpack the Pantry distribution file using **tar -xzf pantry-*.tar.gz**. Change to the newly made directory with **cd pantry-*/**.
2. Become root, and issue **python setup.py install**. That's it!
3. Optional: install manual page. The Pantry installer only installs the Panty source code and two very short scripts used to start Pantry. It does not install the man page. In the Pantry distribution, you will find a file `docs/pantry.1`. Copy this to a location where your other locally installed section 1 manpages are--on my system, that is `usr/local/share/man/man1/`.

   You will also find the manpage in Appendix A.

The installer only installs the Python source code and two trivial launcher scripts. It does not install documentation or many other handy files that are in the Pantry distribution. So you will want to keep handy the directory where you unpacked the Pantry tarball. We'll use some of those files later on.

Speaking of documentation, you will find this manual in many formats (including PDF, plain text, and the XML source code) in the `docs/` directory.

1.2. Possible installation problems

The installer might tell you your version of Python is too old. Pantry requires at least Python 2.4.[2] If your Python is too old, use your distribution's package manager to install a newer one.

1.3. Tailoring your installation

If you want to install Pantry as a non-root user, you'll want to read the Python distutils manual (http://docs.python.org/inst/inst.html), which explains various options for the **setup.py** script.

Also, I should warn you that Python's distutils has no way to uninstall what it installs. If you want to easily remove Pantry, users of most distributions should look into using CheckInstall (http://asic-linux.com.mx/~izto/checkinstall/). Gentooists can easily write an ebuild for Pantry using the distutils eclass.

1.4. About the documentation

The Pantry User Guide, the document that you are reading right now, is available in several formats in the Pantry distribution in the directory named `docs`. There you will find one gigantic HTML file and a plain-text version as well. You will also find an HTML version that has been "chunked" into smaller files in the `docs/html-chunk` directory. Finally, you will also find a PDF version in US Letter-sized paper. Unfortunately, right now I do not know how to make a good PDF in A4 paper, so US Letter is all you will find.

There are many examples sprinkled throughout the text. In these examples, your shell prompt is indicated with `$`. Many of the lines we will be entering in the examples are quite long. When you type them into your computer, you can just enter them on one line. However, this would look bad in the printed documentation, so we use the backslash to continue lines across line breaks. The beginning of each continued line is indicated with a >. Don't enter the > if you are entering the lines at your computer.

Notes

1. Pantry may also be run on Windows, though this is not tested at this point and the documentation is Unix-centric. Frankly I doubt many Windows users would be interested in running Pantry in a Windows environment; Windows is pretty hostile to the command line. Maybe running it in Cygwin (http://www.cygwin.com/) wouldn't be so bad though.
2. Pantry uses at least one Python module, **subprocess**, that is new in Python 2.4.

Chapter 2. Basic Pantry commands

The core program of Pantry is named, appropriately enough, **pantry**. In this section you'll learn how **pantry** works. But first you'll need to understand the way Pantry was designed--that is, the Pantry Paradigm.

2.1. The Pantry Paradigm

The basic unit of Pantry is the *food*. Foods are grouped together and stored on your computer in files. There are six different kinds of files that store foods, but the most important kind is simply called a Pantry native file.[1] Pantry can store thousands of foods in a single Pantry native file, yet Pantry will be able to locate and search for foods very quickly.

Pantry comes with a Pantry native file named `master` that contains 7,294 foods. These foods come from the U.S. Department of Agriculture's National Nutrient Databse for Standard Reference, release 19 (http://www.nal.usda.gov/fnic/foodcomp/search/). A big thank-you goes to these folks for producing this database--without it Pantry would never have been written.[2]

Every food has several *traits*. Later on you will learn how you can set a food's traits; foods in the `master` file already have their traits set. The traits are:

Food traits

`name`

The name of this food, such as *`Bananas, raw`*.

`date`

The date on which you ate this food. This is a string; no special date formatting rules apply to it.

`meal`

The meal in which you ate this food (*`Breakfast Lunch`*, *`midnight snack`*, etc--whatever you wish)

`group`

Useful for grouping foods together. You can use familiar food groups (such as Dairy, Poultry, etc.) or you might group foods together if you eat them together (for instance you might have a group "Cereal and Milk" in which you place those two foods.) We'll learn more about how you can use groups later.

In the `master` file, each food's `group` attribute is already set to one of twenty-four food groups, such as *`Fruits and fruit juices`* or *`Snacks`*.

`qty`

How much of this food you ate. Pantry records this internally as a string; Pantry internally converts it to a number as necessary to perform calculations. You can therefore set the `qty` trait to a string that will convert to an integer or to a floating-point number. This can be a number or a fraction, such as `1/3`. You can set `qty` equal to zero, but this does not delete the food--later we will discuss how to delete foods. You can even use negative numbers, though I do not know why you would want to do such a thing.

`unit`

A description of the amount of this food you ate. The units that you can pick from vary for different foods. We will use the term *available units* to refer to the units that you can pick from for a particular food. Every food has at least three available units: *`oz`*, *`g`*, and *`lb`*. The other available units vary by food. For example, the food in the `master` file named *`Bananas, raw`* has several other units available, including *`Cup, mashed`*; *`large (8" to 8-7/8" long)`*, and *`extra small (less than 6" long)`*.

When you keep track of which foods you eat, you'll set the quantity and unit to whatever makes sense: for instance, if you eat some *`Bananas, raw`*, you can set the quantity to *`2`* and the unit to *`large (8" to 8-7/8" long)`*. Remember, you can use any number for quantity you want, including floating point numbers, but you can set the unit only to what is available for that particular food.

`pctRefuse`

The percent of this food that is waste. For example, with an apple, the core is refuse; for a chicken drumstick, the bone is refuse. Many foods have no refuse; in that case, this trait will be set equal to zero, or to *`None`*. As with the `qty` trait, Pantry internally keeps this as a string, converting it to a number as needed.

`refDesc`

A description of the refuse, such as *`Core`* or *`Bone`*.

`comment`

Whatever notes you may wish to add.

`order`

Any string. As we will see later, **pantry** can sort reports however you like; using the `order` trait, you can sort foods into any arbitrary order.

All Pantry traits are strings. This includes the date trait. Thus, you can set the date trait for a food to *`2007-05-06`*, *`05-06`*, *`Tuesday`*, or even *`Who cares?`* if you wish. You may also set this trait, and all traits except `unit` and `qty`, Only the `qty` and `unit` traits must be set to a non-zero-length value.

Pantry works by copying foods from one file to another. This makes the **pantry** command mostly a glorified copier. **pantry** starts by examining all the foods in each file that you specify. If you do not

specify any `search options`, then all foods in the each file you specify are copied to a temporary place that we will call the *buffer*. Otherwise, if you specify any `search options`, then only the foods whose traits match *all* of the search options will be copied to the buffer. Any food not matching the `search options` you specify is ignored. **pantry** then modifies the traits of the foods in the buffer using any `change options` you specified. **pantry** can then send the buffer to a report (which prints the foods to standard output) and add the buffer to other files. All this is best understood with examples.

2.2. Basic food searches

Because Pantry works by copying and changing foods from food files, you'll first need a food file with interesting foods in it. The Pantry distribution contains a file named `master`. It contains over 7,000 foods, which should be enough to get you started. Copy this file to a convenient place (I'd suggest `~/pantry/master` unless you have a better idea) because you'll be using it often and you might even want to change it. Change to the directory that contains your copy and we'll get started!

Like many Unix commands, the synopsis of the **pantry** command takes zero or more arguments and zero or more options, that is: **pantry [OPTIONS] [FILE ...]** . FILE specifies the file you wish to search for foods. **pantry** will search using OPTIONS that you may specify. OPTIONS also perform many other useful tasks, such as modifying the traits of the foods **pantry** finds, printing reports of the foods **pantry** finds, and adding the resuls to files.

For a simple example, run **`pantry --name Bananas master`** in the same directory as the file containing your `master` file. What happens? Well, as far as you can tell, nothing. But things actually were happening behind the scenes. First, **pantry** examined every food in `master`. Because you specified a search option, `--name Bananas`, Pantry copied all foods matching that criterion into a buffer. However, you did not tell **pantry** to actually do anything with the buffer. So **pantry** terminated without showing you anything at all, and returned you to your command prompt. The buffer that **pantry** made is gone, never to be seen nor heard from again.

2.3. Using the `--print` option

To actually see the buffer, use the `--print` option. It takes a single argument, called a *report*. Here is an example for starters:

Example 2-1. Introducing the --print option

```
$ pantry --name Bananas --print names master
Bananas, raw
Bananas, dehydrated, or banana powder
Cereals ready-to-eat, KELLOGG'S, CORN FLAKES With Real Bananas
```

As you can see, `--print names` simply prints the name of each food in the buffer. Here the name of the report is *`names`*. Other handy reports are *`traits`*, *`units`*, and *`nuts`*, to print the traits, available units,

and nutrient breakdown of each food in the buffer.

The *traits* report only shows a trait if it is not equal to zero or None (in the case of the pctRefuse or qty traits) or if it is not equal to a zero-length string (in the case of any other trait.)

Example 2-2. Using --print traits

```
$ pantry --name Bananas --print traits master
Bananas, raw
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
100 g (100g)
Bananas, dehydrated, or banana powder
Group: Fruits and Fruit Juices
100 g (100g)
Cereals ready-to-eat, KELLOGG'S, CORN FLAKES With Real Bananas
Group: Breakfast Cereals
100 g (100g)
```

The *units* report prints each food's available units. It does not print *g*, *oz*, or *lb* as these are available for every food.

Example 2-3. Using --print units

```
$ pantry --name "Bananas, raw" --print units master
   cup, mashed
   large (8" to 8-7/8" long)
   medium (7" to 7-7/8" long)
   extra small (less than 6" long)
   cup, sliced
   NLEA serving
   small (6" to 6-7/8" long)
   extra large (9" or longer)
```

The *nuts* report prints a food's nutrient breakdown. Later we will talk about why **pantry** printed these particular nutrients, as there are many more nutrients for *Bananas, raw* than were printed here. Also, later we will learn what the two rightmost columns in the report mean (you probably can tell what the first two columns are for.)

Example 2-4. Using --print nuts

```
$ pantry --name "Bananas, raw" --print nuts master
Nutrient                  Amount          %G     %TOT
-------------------------------------------------------
Calories                  89   kcal        4      100
Total Fat                 0    g           1      100
Saturated Fat             0    g           1      100
Cholesterol               0    mg          0      NA
Sodium                    1    mg          0      100
Total Carbohydrate        23   g           8      100
```

```
Dietary Fiber            3    g          10     100
Sugars                   12   g          NG     100
Protein                  1    g          2      100
Vitamin A                64   IU         1      100
Vitamin C                9    mg         14     100
Calcium                  5    mg         1      100
Iron                     0    mg         1      100
```

The *nuts* and *units* reports print only a food's nutrients and available units, respectively. They do not print anything else about the food, not even its `name` trait. When there is more than one food in the buffer, one report is printed for each food. It may be obvious that you are looking at the results for more than one food when you are examining a set of *nuts* reports.

Example 2-5. Two *nuts* reports

```
$ pantry --name Papaya --print names master
Papayas, raw
Papaya nectar, canned
$ pantry --name Papaya --print nuts master
Nutrient                 Amount         %G     %TOT
-----------------------------------------------------
Calories                 39   kcal      2      41
Total Fat                0    g         0      48
Saturated Fat            0    g         0      48
Cholesterol              0    mg        0      NA
Sodium                   3    mg        0      38
Total Carbohydrate       10   g         3      40
Dietary Fiber            2    g         7      75
Sugars                   6    g         NG     30
Protein                  1    g         1      78
Vitamin A                1094 IU        22     75
Vitamin C                62   mg        103    95
Calcium                  24   mg        2      71
Iron                     0    mg        1      23
Nutrient                 Amount         %G     %TOT
-----------------------------------------------------
Calories                 57   kcal      3      59
Total Fat                0    g         0      52
Saturated Fat            0    g         0      52
Cholesterol              0    mg        0      NA
Sodium                   5    mg        0      63
Total Carbohydrate       15   g         5      60
Dietary Fiber            1    g         2      25
Sugars                   14   g         NG     70
Protein                  0    g         0      22
Vitamin A                361  IU        7      25
Vitamin C                3    mg        5      5
Calcium                  10   mg        1      29
Iron                     0    mg        2      77
```

However, in the next example, you cannot even tell that you are looking at two *units* reports here, and you certainly cannot tell the two foods apart. This next example shows units reports for both *Papaya nectar, canned* and *Papayas, raw*:

Example 2-6. Two *units* reports

```
$ pantry --name Papaya --print units master
   small (4-1/2" long x 2-3/4" dia)
   medium (5-1/8" long x 3" dia)
   cup, mashed
   cup, cubes
   large (5-3/4" long x 3-1/4" dia)
   fl oz
   cup
```

We will discover a solution to this problem in the next section.

2.3.1. Combining reports

As you saw above, using just the *nuts* or *units* reports can be confusing, especially when you have more than one food in your buffer, because these reports do not indicate which food goes with which nutrients or with which units.

An easy solution for this is to combine reports. By separating each report name with a dash, you may tell **pantry** to print more than one report for each food.

Example 2-7. Combining reports

```
$ pantry --name Papaya --print names-units master
Papayas, raw
   small (4-1/2" long x 2-3/4" dia)
   medium (5-1/8" long x 3" dia)
   cup, mashed
   cup, cubes
   large (5-3/4" long x 3-1/4" dia)
Papaya nectar, canned
   fl oz
   cup
$ pantry --name Papaya --print traits-nuts master
Papayas, raw
Group: Fruits and Fruit Juices
Refuse: 33 percent Seeds and skin
100 g (100g)
Nutrient                  Amount         %G     %TOT
-----------------------------------------------------
Calories                  39   kcal       2      41
Total Fat                 0    g          0      48
Saturated Fat             0    g          0      48
Cholesterol               0    mg         0      NA
```

```
Sodium                    3    mg          0      38
Total Carbohydrate        10   g           3      40
Dietary Fiber             2    g           7      75
Sugars                    6    g           NG     30
Protein                   1    g           1      78
Vitamin A                 1094 IU          22     75
Vitamin C                 62   mg          103    95
Calcium                   24   mg          2      71
Iron                      0    mg          1      23
Papaya nectar, canned
Group: Fruits and Fruit Juices
100 g (100g)
Nutrient                  Amount          %G     %TOT
-------------------------------------------------------
Calories                  57   kcal        3      59
Total Fat                 0    g           0      52
Saturated Fat             0    g           0      52
Cholesterol               0    mg          0      NA
Sodium                    5    mg          0      63
Total Carbohydrate        15   g           5      60
Dietary Fiber             1    g           2      25
Sugars                    14   g           NG     70
Protein                   0    g           0      22
Vitamin A                 361  IU          7      25
Vitamin C                 3    mg          5      5
Calcium                   10   mg          1      29
Iron                      0    mg          2      77
```

As you can see, some whitespace would help make this more readable. To add whitespace, use the *`blank`* report. It simply prints a blank line.

Example 2-8. Using the *`blank`* report

```
$ pantry --name Papaya --print traits-nuts-blank master
Papayas, raw
Group: Fruits and Fruit Juices
Refuse: 33 percent Seeds and skin
100 g (100g)
Nutrient                  Amount          %G     %TOT
-------------------------------------------------------
Calories                  39   kcal        2      41
Total Fat                 0    g           0      48
Saturated Fat             0    g           0      48
Cholesterol               0    mg          0      NA
Sodium                    3    mg          0      38
Total Carbohydrate        10   g           3      40
Dietary Fiber             2    g           7      75
Sugars                    6    g           NG     30
Protein                   1    g           1      78
Vitamin A                 1094 IU          22     75
Vitamin C                 62   mg          103    95
Calcium                   24   mg          2      71
```

```
Iron                      0    mg          1      23

Papaya nectar, canned
Group: Fruits and Fruit Juices
100 g (100g)
Nutrient                  Amount          %G     %TOT
-------------------------------------------------------
Calories                  57   kcal        3      59
Total Fat                 0    g           0      52
Saturated Fat             0    g           0      52
Cholesterol               0    mg          0      NA
Sodium                    5    mg          0      63
Total Carbohydrate        15   g           5      60
Dietary Fiber             1    g           2      25
Sugars                    14   g           NG     70
Protein                   0    g           0      22
Vitamin A                 361  IU          7      25
Vitamin C                 3    mg          5      5
Calcium                   10   mg          1      29
Iron                      0    mg          2      77
```

2.4. Details on searching

Here are some important details about searching.

2.4.1. More traits you may use to search

As you've seen, `name` is one trait you can search by. When searching, you may limit your buffer by any traits you wish. Here are the options you use in order to limit your search by particular traits. As with many Unix commands, you can use either a long option (indicated by two dashes) or a short option (with one dash).

- `-n` or `--name`
- `-g` or `--group`
- `-d` or `--date`
- `-m` or `--meal`
- `-u` or `--unit`
- `-q` or `--qty`
- `-c` or `--comment`
- `-o` or `--order`

You may use as many of these options in a single **pantry** command as you wish. If you use more than one search option, then Pantry will add foods to the buffer only if they match all the traits you specify:

Example 2-9. Using multiple search options

```
$ pantry --name Bananas --print traits master
Bananas, raw
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
100 g (100g)
Bananas, dehydrated, or banana powder
Group: Fruits and Fruit Juices
100 g (100g)
Cereals ready-to-eat, KELLOGG'S, CORN FLAKES With Real Bananas
Group: Breakfast Cereals
100 g (100g)
$ pantry --name Bananas --group Breakfast --print traits master
Cereals ready-to-eat, KELLOGG'S, CORN FLAKES With Real Bananas
Group: Breakfast Cereals
100 g (100g)
```

2.4.2. Searches are regular expressions

A very important detail about searches is that all search patterns are regular expressions.[3] This allows you to be very flexible in how you specify your searches. You've already seen one consequence of this: a search pattern only needs to match a portion of a trait in order for that food to be included in the results--that is, `--name Bananas` matches foods that have "Bananas" anywhere in their name.

If you don't want to learn regular expressions, that's okay. Simple searches using just letters and numbers will work just fine. However, you should know that the following characters have special meanings in regular expressions:

```
[\^$.|?*+()
```

If you don't know regular expressions, avoid including any of these characters in your search patterns. Remember to quote your search patterns if they include spaces. For example:**`pantry --name 'Bananas, raw' --print traits-units master`**.

If you want to learn more about the power of regular expressions, this website (http://www.regular-expressions.info/) is a great place to start. For those already familiar with regular expressions, following is an example of how they can come in handy. This also demonstrates the power of Pantry's command line interface, as it is easy to use Pantry with other Unix programs such as **wc**, which counts words and lines.

Example 2-10. Using regular expressions

```
$ pantry --name Milk --print names master | wc -l
54
$ pantry --name "Milk.*reduced" --print names master
Milk, chocolate, fluid, commercial, reduced fat, with added calcium
Milk, reduced fat, fluid, 2% milkfat, with added nonfat milk solids and vitamin A
Milk, chocolate, fluid, commercial, reduced fat
Milk, reduced fat, fluid, 2% milkfat, with added vitamin A
Milk, reduced fat, fluid, 2% milkfat, protein fortified, with added vitamin A
Milk, buttermilk, fluid, cultured, reduced fat
Milk, reduced fat, fluid, 2% milkfat, with added nonfat milk solids, without added vitamin A
Milk, dry, nonfat, calcium reduced
```

As the previous example shows if your search pattern includes characters that are special to your shell, remember to quote it--but you already knew that if you're familiar with regular expressions.

2.4.3. My search is returning foods I don't want!

Because search patterns are regular expressions, they will sometimes return more foods than you are interested in. In the next example, I only want to know about *`Apples, raw, without skin`*. However, the buffer also includes two other foods that contain that same string:

Example 2-11. A search that has more foods than I am interested in

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

There are two ways to fix this. The first way is to use a feature of regular expressions called *anchors*. The `$` anchor matches the end of the string. Here, it tells Pantry that there cannot be any characters between *`Apples, raw, without skin`* and the end of the `name` trait. The anchor is only the dollar sign; the backslash is there in order to keep the shell from giving the dollar sign a special meaning. See this webpage (http://www.mpi-inf.mpg.de/~uwe/lehre/unixffb/quoting-guide.html) for more information on why you need to quote characters in Unix shells.[4]

Example 2-12. Using anchors

```
$ pantry --name "Apples, raw, without skin\$" --print names \
> master
Apples, raw, without skin
```

Another way to fix this problem is to use the `--exact-match` or `-x` option. This option changes all `search options` so that they no longer use regular expressions. Instead, with this option, a food's traits must exactly match the search options in order to be included in the buffer.

Example 2-13. Using the `--exact-match` option

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

2.4.4. Searches are case sensitive

By default, all searches in Pantry are case sensitive. This is true both for searches by trait as well as when you are using the `--c-unit` option. In addition, searches using the `--exact-match` option are also case sensitive. To make all these searches case insensitive, use the `--ignore-case` or `-i` option.

2.5. Using nutrient lists to compare foods to particular nutrient goals

2.5.1. What are nutrient lists?

In Pantry, *nutrient lists* have two purposes: first, they allow you to select which nutrients appear in *`nuts`* reports; and second, they allow you to compare your nutrient intake to certain goals.

If you're keeping track of your food intake, you'll probably want to know how your nutrient intake compares against certain goals you've set for yourself. For instance, you might decide you want to take in 65 grams of fat every day, or 2800 calories a day. You can tell Pantry what your goals are and it will help you compare certain foods, or all your food intake if you wish, against those goals.

In addition, the `master` in Pantry has dozens of nutrients. Some of them, such as *`Calories`* or *`Vitamin A`*, are quite familiar. Others, such as *`18:0`*, probably don't interest you unless you're a scientist. Using nutrient lists, you decide which nutrients appear in your *`nuts`* reports, and the order in which they appear.

As the name suggests, a nutrient list includes multiple nutrients, in a specific order. Each nutrient is specified using both its name and its units: for example, *`Protein`* and *`g`*, or *`Calories`* and *`kcal`*. When Pantry prints a *`nuts`* report, it prints only the nutrients included in the nutrient list.

In addition, each nutrient in the nutrient list optionally has a *goal* associated with it. For exaample, if you wish to consume 2800 calories a day, you can set the goal for calories to 2800. If you do not specify a

goal, but you include a nutrient in a nutrient list, **pantry** will still print the nutrient in *nuts* reports. If you don't want a nutrient shown in your *nuts* reports, then do not include the nutrient in your nutrient list. We are getting ahead of ourselves a bit, though. Later we will discuss how you can define your own nutrient lists. For now, though, you are stuck with the nutrient lists that are included with Pantry:

Nutrient lists included with Pantry

facts

This nutrient list mimics the USA "Nutrition Facts" panel; that is, it shows nutrients in the same order as they appear on those labels, and the nutrient goals are identical to the FDA Daily Values. One of the nutrients that appears on the "Nutrition Facts" panel, *Sugars*, has no FDA Daily Value, so there is no goal for this nutrient.

all

Shows every possible nutrient. It does not, however, include any nutrient goals.

short

Shows only *Calories*, *Total Fat*, *Total Carbohydrate*, and *Protein*. As with *all*, this nutrient list does not include any nutrient goals.

dv

Includes, in alphabetical order by nutrient name, every nutrient for which there is an FDA Daily Value; the goals are the respective Daily Values.

2.5.2. How nutrient lists are displayed in *nuts* reports

As you've seen, the *nuts* report has four columns. The first two columns show the nutrient name and the amount of the nutrient, respectively. The third column shows the nutrient's percentage of the any goal that has been set for this nutrient in the nutrient list. The fourth column shows this nutrient's percentage of the total amount of this nutrient for the buffer.

By default, Pantry uses the *facts* nutrient list. Therefore, in the next example, we see that 100 grams of bananas has 89 calories. The *facts* nutrient list has a goal of 2000 calories, the same amount used for the "Nutrition Facts" labels. Therefore, the report shows four percent in the third column. Other nutrients show similar results, except for *Sugars*. This shows *NG*, for No Goal. This is because although the "Nutrition Facts" panel shows *Sugars*, there is no FDA Daily Value for this nutrient. Accordingly, the *facts* nutrient list has no goal for this nutrient either.

Because there is only one food in the buffer, every value except one in the last column is 100 percent. The value for Cholesterol is *NA* because the buffer has no cholesterol.

Example 2-14. The *facts* nutrient list with 100 grams of bananas

```
$ pantry --name "Bananas, raw" --print traits-nuts master
```

```
Bananas, raw
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
100 g (100g)
Nutrient                  Amount         %G     %TOT
-------------------------------------------------------
Calories                  89   kcal       4      100
Total Fat                 0    g          1      100
Saturated Fat             0    g          1      100
Cholesterol               0    mg         0      NA
Sodium                    1    mg         0      100
Total Carbohydrate        23   g          8      100
Dietary Fiber             3    g          10     100
Sugars                    12   g          NG     100
Protein                   1    g          2      100
Vitamin A                 64   IU         1      100
Vitamin C                 9    mg         14     100
Calcium                   5    mg         1      100
Iron                      0    mg         1      100
```

2.5.3. Using a different nutrient list

By default, Pantry uses the *facts* nutrient list when printing reports. You may specify a different nutrient list with the `--nutrient-list` option. In the next example, the third column always shows `NG` because the *short* nutrient list includes no goals.

Example 2-15. Specifying a nutrient list

```
$ pantry --name "Bananas, raw" --print traits-nuts --nutrient- \
> list short master
Bananas, raw
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
100 g (100g)
Nutrient                  Amount         %G     %TOT
-------------------------------------------------------
Calories                  89   kcal       NG     100
Total Fat                 0    g          NG     100
Total Carbohydrate        23   g          NG     100
Protein                   1    g          NG     100
```

2.6. Using summary reports to see information about an entire buffer

So far, all the reports you have seen print one report for each food. For instance, the *names* report prints the name of each food in the buffer, and the *nuts* report prints one *nuts* report for each food in the buffer.

Sometimes you will want to know information about an entire buffer, rather than about each food in the buffer. Two reports provide information about the entire buffer: *sum* and *groups*.

2.6.1. Using the *sum* report

The *sum* report prints information about the sum of all nutrients of every food in the buffer. Like *nuts* reports, it uses nutrient lists to determine which nutrients appear in the report and which goals to use. It looks almost identical to the *nuts* report, with one difference: it has only three columns, rather than four. The first column shows the nutrient name; the second shows the nutrient amount; and the third shows the amount's percentage of a goal defined in the nutrient list.

As with the reports we discussed earlier, you may combine summary reports with other reports, as the next example demonstrates by combining a *traits* report and a *sum* report. When you combine summary reports and food reports, the summary reports are always shown last.

Example 2-16. Using the *sum* report

```
$ pantry --name Bananas --print traits-blank-sum master
Bananas, raw
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
100 g (100g)

Bananas, dehydrated, or banana powder
Group: Fruits and Fruit Juices
100 g (100g)

Cereals ready-to-eat, KELLOGG'S, CORN FLAKES With Real Bananas
Group: Breakfast Cereals
100 g (100g)

SUM:
Nutrient                  Amount         %G
-----------------------------------------------
Calories                  852  kcal       43
Total Fat                 11   g          17
Saturated Fat             8    g          42
Cholesterol               0    mg         0
Sodium                    456  mg         19
```

```
Total Carbohydrate        195  g          65
Dietary Fiber             15   g          60
Sugars                    85   g          NG
Protein                   8    g          16
Vitamin A                 1983 IU         40
Vitamin C                 43   mg         71
Calcium                   56   mg         6
Iron                      7    mg         41
```

As with *nuts* reports, you use nutrient lists to determine which nutrients are shown in the *sum* report, and what the goals are.

Example 2-17. Using the `--nutrient-list` option with the *sum* report

```
$ pantry --name Bananas --print traits-blank-sum --nutrient-list short \
> master
Bananas, raw
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
100 g (100g)

Bananas, dehydrated, or banana powder
Group: Fruits and Fruit Juices
100 g (100g)

Cereals ready-to-eat, KELLOGG'S, CORN FLAKES With Real Bananas
Group: Breakfast Cereals
100 g (100g)

SUM:
Nutrient                  Amount         %G
-------------------------------------------------
Calories                  852  kcal       NG
Total Fat                 11   g          NG
Total Carbohydrate        195  g          NG
Protein                   8    g          NG
```

2.6.2. Using the *groups* reports

The other summary report is the *groups* report. It prints a list of all the groups in the buffer, the number of foods in each, and the total number of foods and groups. For instance, you can run the following example to see how many groupsa and foods are in the `master` file. Remember that if you do not specify any search options, **pantry** copies all foods from the files you specify into the buffer, which is why this example shows you all the groups in the `master` file.

Example 2-18. How many groups are in the `master` file?

```
$ pantry --print groups master
  Group                                No. of Foods
-------------------------------------------------------------
  Baby Foods                                289
  Baked Products                            488
  Beef Products                             783
  Beverages                                 266
  Breakfast Cereals                         427
  Cereal Grains and Pasta                   169
  Dairy and Egg Products                    216
  Ethnic Foods                              132
  Fast Foods                                310
  Fats and Oils                             239
  Finfish and Shellfish Products            255
  Fruits and Fruit Juices                   314
  Lamb, Veal, and Game Products             343
  Legumes and Legume Products               234
  Meals, Entrees, and Sidedishes             99
  Nut and Seed Products                     128
  Pork Products                             294
  Poultry Products                          346
  Sausages and Luncheon Meats               232
  Snacks                                    131
  Soups, Sauces, and Gravies                399
  Spices and Herbs                           60
  Sweets                                    351
  Vegetables and Vegetable Products         789
-------------------------------------------------------------
                                           7294 foods total
                                             24 groups total
```

2.7. Changing food traits

Pantry would not be very useful if you could only search the `master` for its contents and print results from it. Fortunately, Pantry makes it very easy for you to change the traits of foods.

When you run **pantry** with the search options we discussed above, **pantry** first searches the files you specified for foods with the traits you specified with your options. **pantry** then changes the resulting foods to the traits you specify, as we will discuss next. For instance, in the next example we change the *`date`* trait of a raw banana. However, as the example also shows, the change is only temporary. Only the food in the buffer is changed, and the food in the buffer is a copy of the food in the `master` file. Later we will learn how to save the new foods that you make.

Example 2-19. Changing foods

```
$ pantry --name "Bananas, raw" --print traits master
Bananas, raw
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
100 g (100g)
$ pantry --name "Bananas, raw" --c-date "2007-05-06" \
> --print traits master
Bananas, raw
Group: Fruits and Fruit Juices
Date: 2007-05-06
Refuse: 36 percent Skin
100 g (100g)
$ pantry --name "Bananas, raw" --print traits master
Bananas, raw
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
100 g (100g)
```

There are eight options you use to change food traits. Each takes an argument to indicate what you would like to change the trait to:

- `--c-name` or `-N`
- `--c-group` or `-G`
- `--c-date` or `-D`
- `--c-meal` or `-M`
- `--c-comment` or `-C`
- `--c-unit` or `-U`
- `--c-qty` or `-Q`
- `--c-order` or `-O`

Changing the `name`, `group`, `date`, `meal`, and `comment` traits is easy. You can change these to any string that you wish, including a zero-length string. This gives you a great deal of flexibility. The `option` trait need not be set to a "official" food group. The `date` trait does not need to respect any particular date format--indeed, strictly speaking, it need not be any date at all!

Only two traits are not this flexible. As with the other traits, Pantry stores the `qty` trait as a string, but it is converted internally to a number as needed, so the value of this trait must be something that can be converted. It can be an integer, a floating-point number, or even a fraction or mixed number. Entering zero does *not* delete foods; later we'll discuss how to delete foods.

The other inflexible trait is the `unit` trait. This trait must be equal to one of the available units for each particular food. The argument that the `--c-unit` takes is actually a regular expression. **pantry** searches the food's available units for a single unit matching that regular expression.

Example 2-20. Changing the unit trait

```
$ pantry --name "Bananas, raw" --c-date "May 7" \
> --c-qty 2 --c-unit medium --print traits-nuts master
Bananas, raw
Group: Fruits and Fruit Juices
Date: May 7
Refuse: 36 percent Skin
2 medium (7" to 7-7/8" long) (236g)
Nutrient                  Amount         %G     %TOT
-----------------------------------------------------
Calories                  210  kcal       11     100
Total Fat                 1    g          1      100
Saturated Fat             0    g          1      100
Cholesterol               0    mg         0      NA
Sodium                    2    mg         0      100
Total Carbohydrate        54   g          18     100
Dietary Fiber             6    g          25     100
Sugars                    29   g          NG     100
Protein                   3    g          5      100
Vitamin A                 151  IU         3      100
Vitamin C                 21   mg         34     100
Calcium                   12   mg         1      100
Iron                      1    mg         3      100
```

If the regular expression you enter as an argument for `--c-unit` matches more than one of a food's available units, **pantry** will give you a warning message. **pantry** will not include the food (either changed or unchanged) in the search results; therefore, the food will not be printed in any reports you may have asked for using the `--print` option. However, Pantry will still include in the search results other foods that it did change. In the next example, the food *`Bananas, raw`* cannot have its units changed, because the search string *`cup`* matches more than one of the food's available units. However, the food *`Bananas, dehydrated, or banana powder`* is included in the search results because its unit was successfully changed.

Example 2-21. Errors when changing the unit trait

```
$ pantry --name "Bananas," --print traits-units master
Bananas, raw
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
100 g (100g)
   cup, mashed
   large (8" to 8-7/8" long)
   medium (7" to 7-7/8" long)
   extra small (less than 6" long)
   cup, sliced
   NLEA serving
   small (6" to 6-7/8" long)
   extra large (9" or longer)
Bananas, dehydrated, or banana powder
Group: Fruits and Fruit Juices
100 g (100g)
```

```
   tbsp
   cup
$ pantry --name "Bananas," --c-unit cup --print traits-units \
> master
==========
pantry-dev: warning: food Bananas, raw will not be copied into the buffer.
Failed to set units using pattern cup
Matches:
cup, mashed
cup, sliced
==========
Bananas, dehydrated, or banana powder
Group: Fruits and Fruit Juices
100 cup (10000g)
   tbsp
   cup
```

One nicety is that you can use fractions and mixed numbers for the `qty` trait:

Example 2-22. `qty` may be a fraction or mixed number

```
$ pantry --name "Bananas, raw" --c-qty '1/2' \
> --c-unit medium --print traits-nuts master
Bananas, raw
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
1/2 medium (7" to 7-7/8" long) (59g)
Nutrient                  Amount         %G     %TOT
-----------------------------------------------------
Calories                  53   kcal       3      100
Total Fat                 0    g          0      100
Saturated Fat             0    g          0      100
Cholesterol               0    mg         0      NA
Sodium                    1    mg         0      100
Total Carbohydrate        13   g          4      100
Dietary Fiber             2    g          6      100
Sugars                    7    g          NG     100
Protein                   1    g          1      100
Vitamin A                 38   IU         1      100
Vitamin C                 5    mg         9      100
Calcium                   3    mg         0      100
Iron                      0    mg         1      100
$ pantry --name "Bananas, raw" --c-qty '2 1/2' \
> --c-unit medium --print traits-nuts master
Bananas, raw
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
2 1/2 medium (7" to 7-7/8" long) (295g)
Nutrient                  Amount         %G     %TOT
-----------------------------------------------------
Calories                  263  kcal       13     100
Total Fat                 1    g          1      100
```

```
Saturated Fat             0    g          2      100
Cholesterol               0    mg         0      NA
Sodium                    3    mg         0      100
Total Carbohydrate        67   g          22     100
Dietary Fiber             8    g          31     100
Sugars                    36   g          NG     100
Protein                   3    g          6      100
Vitamin A                 189  IU         4      100
Vitamin C                 26   mg         43     100
Calcium                   15   mg         1      100
Iron                      1    mg         4      100
```

You can use fractions or mixed numbers anywhere that Pantry expects to get a number. That includes all command-line options as well as within XML files, which we will discuss later.

2.8. Saving new foods with `--add` and `--edit`; deleting with the `--delete` option

You'll probably want to store the new foods you make. You'll usually do this using the `--add` or `-a` option. This option takes a single *`FILENAME`* argument. *`FILENAME`* can be almost any name you want. For now, though, it should not end in `.xml`, `.zip`, or `.txt`--we'll learn about what these extensions do later. If *`FILENAME`* does not already exist, **pantry** will create it for you; if the file already exists, the foods returned from your search will be added to the file.

Two identical foods cannot exist in a single file. Two foods are identical if all their traits are identical. When you add a food to a file and there is already an identical food in the file, Pantry will alter the comment trait of the food you are adding by appending *`(Copy 1)`*, *`(Copy 2)`*, etc.

Example 2-23. Using the --add option

```
$ pantry --name "Bananas, raw" --add newfile --print traits \
> master
Bananas, raw
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
100 g (100g)
$ pantry --print traits newfile
Bananas, raw
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
100 g (100g)
```

Another way to store the changes you make is by using the `--edit` option. `--edit` searches for foods and makes changes that you specify. Unlike `--add`, `--edit` does not add the changed foods to a new file; instead, it changes already existing foods, in the same file.

Example 2-24. Using the --edit option

```
$ pantry --name "Bananas, raw" --c-name "Bananas, big \
> yellow" --edit --print traits master
Bananas, big yellow
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
100 g (100g)
$ pantry --name Bananas --print traits-blank master
Cereals ready-to-eat, KELLOGG'S, CORN FLAKES With Real Bananas
Group: Breakfast Cereals
100 g (100g)

Bananas, dehydrated, or banana powder
Group: Fruits and Fruit Juices
100 g (100g)

Bananas, big yellow
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
100 g (100g)
```

Finally, you may delete foods from the original files with the `--delete` option.

Example 2-25. Using the --delete option

```
$ pantry --name "Bananas, big yellow" --print traits --delete \
> master
Bananas, big yellow
Group: Fruits and Fruit Juices
Refuse: 36 percent Skin
100 g (100g)
$ pantry --name "Bananas, big yellow" --print traits master
```

Notes

1. You'll notice if you run the Unix program **file** on a Pantry native file, it is most likely a Berkeley DB file. Unix geeks will cringe at the fact that this file is *not* plain text.
2. Most food databases that you will find anywhere, such as the one at NutritionData.com (http://www.nutritiondata.com), are derived from the same USDA database.
3. Regular expression gurus will appreciate knowing that Pantry uses Perl-compatible regular expressions. Links from this page (http://www.regular-expressions.info/tools.html) will help give you an idea of what this means. If you are familiar with the Unix command-line utilities **grep** and **egrep**, the regular expressions used in Pantry are most similar to those used in **egrep**.
4. An easy solution in this example would have been to use single quotes, but due to idiosyncracies in the scripts that automatically build the examples for the documentation, all the examples in the user guide use double quotes instead.

Chapter 3. Using Pantry: practical examples

Now that you know all the basic Pantry options, you're ready to combine them and put them to use! This chapter will show you how.

3.1. Finding nutrient amounts

Often you will wish to know the nutrient makeup of a particular food. Suppose for example that you want to know how many calories are in a fresh apple. You figure that `master` already has such a food, but you don't know exactly what it would be. So first you search for `apple` to see what comes up.

Example 3-1. Searching for apples

```
$ pantry --ignore-case --name apple --print names master
Apple cider-flavored drink, powder, low calorie, with vitamin C, prepared
English muffins, raisin-cinnamon (includes apple-cinnamon)
Babyfood, cereal, high protein, with apple and orange, prepared with whole milk
Applesauce, canned, unsweetened, without added ascorbic acid
Pineapple, canned, light syrup pack, solids and liquids
Babyfood, fruit, bananas and pineapple with tapioca, strained
Babyfood, fruit, apple and raspberry,  junior
Apple juice, frozen concentrate, unsweetened, undiluted, with added ascorbic acid
Babyfood, apple-banana juice
Fruit salad, (peach and pear and apricot and pineapple and cherry), canned, extra heavy syrup, solids
Babyfood, fruit, apple and raspberry, strained
Pie fillings, apple, canned
Pineapple, frozen, chunks, sweetened
Scrapple, pork
Babyfood, fruit, bananas with apples and pears, strained
[ trimmed to save space ]
```

It turns out that this search returns 163 results. There must be a way to narrow this down. Well, many foods will have the word "Apple" in their name, including many foods that are not even fruits (desserts, for example). A good way to narrow your results is to use the `--group` option:

Example 3-2. Searching for apples, with the `--group` option

```
$ pantry --ignore-case --name apple --group fruits --print names master
Apples, dried, sulfured, uncooked
Apple juice, frozen concentrate, unsweetened, undiluted, with added ascorbic acid
Pineapple, canned, water pack, solids and liquids
Apples, dried, sulfured, stewed, without added sugar
Apple juice, frozen concentrate, unsweetened, undiluted, without added ascorbic acid
Apples, raw, without skin, cooked, boiled
Apple juice, frozen concentrate, unsweetened, diluted with 3 volume water, with added ascorbic acid
Sugar-apples, (sweetsop), raw
Applesauce, canned, unsweetened, without added ascorbic acid
```

```
Pineapple, raw, traditional varieties
Pineapple, canned, light syrup pack, solids and liquids
Applesauce, canned, sweetened, with salt
Fruit salad, (peach and pear and apricot and pineapple and cherry), canned, extra heavy syrup, solids
Pineapple, canned, heavy syrup pack, solids and liquids
Crabapples, raw
[ trimmed to save space ]
```

That is still a lot more than we are looking for, including a bunch of things that merely have "Apple" in their name. What if we use a regular expression to limit the results only to foods whose *name* trait begins with *apple*?

Example 3-3. Searching for apples using a regular expression

```
$ pantry --ignore-case --name ^apple --group fruits --print names master
Apples, dried, sulfured, uncooked
Apples, dried, sulfured, stewed, without added sugar
Apple juice, frozen concentrate, unsweetened, undiluted, without added ascorbic acid
Apples, raw, without skin, cooked, boiled
Apple juice, frozen concentrate, unsweetened, diluted with 3 volume water, with added ascorbic acid
Applesauce, canned, unsweetened, without added ascorbic acid
Apples, dried, sulfured, stewed, with added sugar
Apple juice, frozen concentrate, unsweetened, undiluted, with added ascorbic acid
Applesauce, canned, sweetened, with salt
Apple juice, canned or bottled, unsweetened, with added ascorbic acid
Applesauce, canned, unsweetened, with added ascorbic acid
Applesauce, canned, sweetened, without salt
Apples, raw, with skin
Apple juice, canned or bottled, unsweetened, without added ascorbic acid
Apples, frozen, unsweetened, heated
Apples, raw, without skin
Apples, dehydrated (low moisture), sulfured, uncooked
Apples, raw, without skin, cooked, microwave
Apples, frozen, unsweetened, unheated
Apple juice, frozen concentrate, unsweetened, diluted with 3 volume water without added ascorbic acid
Apples, canned, sweetened, sliced, drained, heated
Apples, dehydrated (low moisture), sulfured, stewed
Apples, canned, sweetened, sliced, drained, unheated
```

That buffer is manageable enough to scan to see what we are looking for. Looks like *Apples, raw, with skin* is what we are looking for. We want to know how many calories are in an apple. To do that, first we need to see what units are available for *Apples, raw, with skin*:

Example 3-4. Available units for an apple

```
$ pantry --name "Apples, raw, with skin" --print names-units \
> master
Apples, raw, with skin
   cup, quartered or chopped
   large (3-1/4" dia) (approx 2 per lb)
   NLEA serving
```

```
medium (2-3/4" dia) (approx 3 per lb)
cup slices
small (2-1/2" dia) (approx 4 per lb)
```

Notice how we combined the *names* and *units* reports rather than using just a *units* report. This is a good idea because otherwise, your report might actually contain results for more than one food, but you would not know this if you used only a *units* report.

Finally, we use the change options to take the apple from the `master` file, change it to the characteristics we're interested in, and print a report:

Example 3-5. How many calories are in an apple?

```
$ pantry --name "Apples, raw, with skin" --c-unit large \
> --c-qty 1 --print traits-nuts master
Apples, raw, with skin
Group: Fruits and Fruit Juices
Refuse: 8 percent Core and stem
1 large (3-1/4" dia) (approx 2 per lb) (212g)
Nutrient                  Amount         %G     %TOT
-------------------------------------------------------
Calories                  110  kcal       6      100
Total Fat                 0    g          1      100
Saturated Fat             0    g          0      100
Cholesterol               0    mg         0      NA
Sodium                    2    mg         0      100
Total Carbohydrate        29   g          10     100
Dietary Fiber             5    g          20     100
Sugars                    22   g          NG     100
Protein                   1    g          1      100
Vitamin A                 114  IU         2      100
Vitamin C                 10   mg         16     100
Calcium                   13   mg         1      100
Iron                      0    mg         1      100
```

3.2. Keeping a food diary

Suppose that on May 8 you want to keep track of everything you eat. Following our previous example, you locate the foods you need in `master`, and then you add them to a file as you'll see in the following example. You print the traits of the foods as you add them to make sure that you're adding the right foods.

Example 3-6. Adding entries to a diary

```
$ pantry --ignore-case --name "apples, raw, with skin" --c-qty \
> 1 --c-unit large --c-date "May 8" --c-meal Breakfast --print \
> traits --add diary master
Apples, raw, with skin
```

```
Group: Fruits and Fruit Juices
Date: May 8 Meal: Breakfast
Refuse: 8 percent Core and stem
1 large (3-1/4" dia) (approx 2 per lb) (212g)
$ pantry --ignore-case --name "kellogg's corn flakes" \
> --c-qty 1 --c-unit cup --c-date "May 8" --c-meal Breakfast \
> --print traits --add diary master
Cereals ready-to-eat, KELLOGG, KELLOGG'S Corn Flakes
Group: Breakfast Cereals
Date: May 8 Meal: Breakfast
1 cup (1 NLEA serving) (28g)
$ pantry --ignore-case --name "milk, reduced fat, fluid, 2% \
> milkfat, with added vitamin A" --c-qty 1 --c-unit cup --c-date \
> "May 8" --c-meal Breakfast --print traits --add diary master
Milk, reduced fat, fluid, 2% milkfat, with added vitamin A
Group: Dairy and Egg Products
Date: May 8 Meal: Breakfast
1 cup (244g)
$ pantry --ignore-case --name "carrots, raw" --c-qty 1 \
> --c-unit ^large --c-date "May 8" --c-meal Lunch --print \
> traits --add diary master
Carrots, raw
Group: Vegetables and Vegetable Products
Date: May 8 Meal: Lunch
Refuse: 11 percent Crown, tops and scrapings
1 large (7-1/4" to 8-/1/2" long) (72g)
$ pantry --ignore-case --name 'chicken.*wing, meat and skin, \
> cooked, roasted' --c-date "May 8" --c-unit wing --c-qty \
> 3 --c-meal Lunch --print traits --add diary master
Chicken, broilers or fryers, wing, meat and skin, cooked, roasted
Group: Poultry Products
Date: May 8 Meal: Lunch
Refuse: 48 percent Bone
3 wing, bone removed (102g)
$ pantry --ignore-case --name "ice creams, chocolate, rich" \
> --c-qty 1 --c-unit cup --c-date "May 8" --c-meal \
> "Lunch" --print traits --add diary master
Ice creams, chocolate, rich
Group: Sweets
Date: May 8 Meal: Lunch
1 cup (148g)
$ pantry --ignore-case --name "mcdonald's, cheeseburger" \
> --c-unit item --c-qty 1 --c-date "May 8" --c-meal \
> "Dinner" --print traits --add diary master
McDONALD'S, Cheeseburger
Group: Fast Foods
Date: May 8 Meal: Dinner
1 item (119g)
$ pantry --ignore-case --name "mcdonald's, french fries" \
> --c-qty 1 --c-unit large --c-date "May 8" --c-meal Dinner \
> --print traits --add diary master
McDONALD'S, French Fries
Group: Fast Foods
```

```
Date: May 8 Meal: Dinner
1 large serving (170g)
$ pantry --ignore-case --name "popcorn, oil-popped, unsalted" \
> --c-qty 2 --c-unit oz --c-date "May 8" --c-meal Dinner \
> --print traits --add diary master
Snacks, popcorn, oil-popped, unsalted
Group: Snacks
Date: May 8 Meal: Dinner
2 oz (57g)
```

Now you want to know a little about what you ate:

Example 3-7. Printing nutrient reports about a diary

```
$ pantry --print traits-nuts-blank diary
Apples, raw, with skin
Group: Fruits and Fruit Juices
Date: May 8 Meal: Breakfast
Refuse: 8 percent Core and stem
1 large (3-1/4" dia) (approx 2 per lb) (212g)
Nutrient                  Amount         %G     %TOT
-------------------------------------------------------
Calories                  110  kcal       6      5
Total Fat                 0    g          1      0
Saturated Fat             0    g          0      0
Cholesterol               0    mg         0      0
Sodium                    2    mg         0      0
Total Carbohydrate        29   g          10     12
Dietary Fiber             5    g          20     22
Sugars                    22   g          NG     30
Protein                   1    g          1      1
Vitamin A                 114  IU         2      1
Vitamin C                 10   mg         16     32
Calcium                   13   mg         1      2
Iron                      0    mg         1      1

Milk, reduced fat, fluid, 2% milkfat, with added vitamin A
Group: Dairy and Egg Products
Date: May 8 Meal: Breakfast
1 cup (244g)
Nutrient                  Amount         %G     %TOT
-------------------------------------------------------
Calories                  122  kcal       6      6
Total Fat                 5    g          7      4
Saturated Fat             3    g          15     8
Cholesterol               20   mg         7      8
Sodium                    100  mg         4      6
Total Carbohydrate        11   g          4      5
Dietary Fiber             0    g          0      0
Sugars                    12   g          NG     17
Protein                   8    g          16     11
Vitamin A                 461  IU         9      3
```

```
Vitamin C                 0    mg          1      2
Calcium                   285  mg          29     37
Iron                      0    mg          0      0

McDONALD'S, French Fries
Group: Fast Foods
Date: May 8 Meal: Dinner
1 large serving (170g)
Nutrient                  Amount          %G     %TOT
-------------------------------------------------------
Calories                  573  kcal        29     26
Total Fat                 30   g           47     27
Saturated Fat             6    g           30     16
Cholesterol               0    mg          0      0
Sodium                    330  mg          14     21
Total Carbohydrate        70   g           23     29
Dietary Fiber             7    g           28     30
Sugars                    0    g           NG     0
Protein                   6    g           11     8
Vitamin A                 0    IU          0      0
Vitamin C                 8    mg          14     27
Calcium                   27   mg          3      3
Iron                      2    mg          10     10

Ice creams, chocolate, rich
Group: Sweets
Date: May 8 Meal: Lunch
1 cup (148g)
Nutrient                  Amount          %G     %TOT
-------------------------------------------------------
Calories                  377  kcal        19     17
Total Fat                 25   g           39     23
Saturated Fat             15   g           77     40
Cholesterol               89   mg          30     38
Sodium                    84   mg          4      5
Total Carbohydrate        31   g           10     13
Dietary Fiber             1    g           5      6
Sugars                    26   g           NG     35
Protein                   7    g           14     10
Vitamin A                 1055 IU          21     7
Vitamin C                 1    mg          1      2
Calcium                   210  mg          21     27
Iron                      2    mg          8      9

Chicken, broilers or fryers, wing, meat and skin, cooked, roasted
Group: Poultry Products
Date: May 8 Meal: Lunch
Refuse: 48 percent Bone
3 wing, bone removed (102g)
Nutrient                  Amount          %G     %TOT
-------------------------------------------------------
Calories                  296  kcal        15     13
Total Fat                 20   g           31     18
```

```
Saturated Fat             6    g          28     15
Cholesterol               86   mg         29     36
Sodium                    84   mg         3      5
Total Carbohydrate        0    g          0      0
Dietary Fiber             0    g          0      0
Sugars                    0    g          NG     0
Protein                   27   g          55     38
Vitamin A                 161  IU         3      1
Vitamin C                 0    mg         0      0
Calcium                   15   mg         2      2
Iron                      1    mg         7      7

Snacks, popcorn, oil-popped, unsalted
Group: Snacks
Date: May 8 Meal: Dinner
2 oz (57g)
Nutrient                  Amount         %G     %TOT
-------------------------------------------------------
Calories                  295  kcal       15     13
Total Fat                 16   g          25     14
Saturated Fat             3    g          14     7
Cholesterol               0    mg         0      0
Sodium                    2    mg         0      0
Total Carbohydrate        33   g          11     14
Dietary Fiber             6    g          23     25
Sugars                    0    g          NG     0
Protein                   5    g          10     7
Vitamin A                 87   IU         2      1
Vitamin C                 0    mg         0      1
Calcium                   6    mg         1      1
Iron                      2    mg         9      9

Carrots, raw
Group: Vegetables and Vegetable Products
Date: May 8 Meal: Lunch
Refuse: 11 percent Crown, tops and scrapings
1 large (7-1/4" to 8-/1/2" long) (72g)
Nutrient                  Amount         %G     %TOT
-------------------------------------------------------
Calories                  30   kcal       1      1
Total Fat                 0    g          0      0
Saturated Fat             0    g          0      0
Cholesterol               0    mg         0      0
Sodium                    50   mg         2      3
Total Carbohydrate        7    g          2      3
Dietary Fiber             2    g          8      9
Sugars                    3    g          NG     5
Protein                   1    g          1      1
Vitamin A                 12104 IU         242    82
Vitamin C                 4    mg         7      14
Calcium                   24   mg         2      3
Iron                      0    mg         1      1
```

```
Cereals ready-to-eat, KELLOGG, KELLOGG'S Corn Flakes
Group: Breakfast Cereals
Date: May 8 Meal: Breakfast
1 cup (1 NLEA serving) (28g)
Nutrient                  Amount         %G     %TOT
-----------------------------------------------------
Calories                  101  kcal       5      5
Total Fat                 0    g          0      0
Saturated Fat             0    g          0      0
Cholesterol               0    mg         0      0
Sodium                    202  mg         8      13
Total Carbohydrate        24   g          8      10
Dietary Fiber             1    g          3      3
Sugars                    3    g          NG     4
Protein                   2    g          4      3
Vitamin A                 501  IU         10     3
Vitamin C                 6    mg         10     20
Calcium                   1    mg         0      0
Iron                      8    mg         45     46

McDONALD'S, Cheeseburger
Group: Fast Foods
Date: May 8 Meal: Dinner
1 item (119g)
Nutrient                  Amount         %G     %TOT
-----------------------------------------------------
Calories                  313  kcal       16     14
Total Fat                 14   g          22     13
Saturated Fat             5    g          26     14
Cholesterol               42   mg         14     18
Sodium                    745  mg         31     47
Total Carbohydrate        33   g          11     14
Dietary Fiber             1    g          5      6
Sugars                    7    g          NG     10
Protein                   15   g          31     21
Vitamin A                 289  IU         6      2
Vitamin C                 1    mg         1      2
Calcium                   199  mg         20     25
Iron                      3    mg         16     16
```

You also want to know what all this adds up to:

Example 3-8. Getting totals about a diary

```
$ pantry --print sum diary
SUM:
Nutrient                  Amount         %G
-----------------------------------------------
Calories                  2217 kcal       111
Total Fat                 111  g          170
Saturated Fat             38   g          191
```

```
Cholesterol              236  mg         79
Sodium                   1599 mg         67
Total Carbohydrate       239  g          80
Dietary Fiber            23   g          92
Sugars                   74   g          NG
Protein                  72   g          144
Vitamin A                14774 IU         295
Vitamin C                31   mg         51
Calcium                  780  mg         78
Iron                     18   mg         98
```

If your diary file contained foods for several days, you would have used `--date` to limit the search results to just today.

If this seemed to require a lot of typing, that's because it did require a lot of typing. Later we'll learn some ways you can speed this process up.

3.3. Organizing your foods into files

3.3.1. A `quick` file

As you've seen from our numerous examples, sometimes the names of foods can get fairly long. Furthermore, because the good people at USDA are so industrious, a search for something seemingly simple in the `master` file can turn up numerous results. For instance, **`pantry --ignore-case --name milk --group dairy --print names master`** returns 64 foods. You probably don't want to type *`Milk, reduced fat, fluid, 2% milkfat, with added vitamin A`* every time you have some milk.

One excellent solution for this problem is to keep copies of foods you eat frequently in a separate, smaller file--perhaps `quick`. If you know that there is only one food in your `quick` file that contains *`milk`*, you can just search for **`pantry --ignore-case --name milk quick`** instead. You can even change the names of foods using `--c-name`, so *`Milk, reduced fat, 2% milkfat, with added vitamin A`* can become simply *`Milk 2%`* if that's what you want.

To make this really easy, change the foods in your `quick` so that the quantity and unit traits are already set to those you use most frequently.

You could also make changes to the `master` file itself, but I prefer to leave this file untouched and store my frequently-eaten foods in a different file.

3.3.2. Organizing diary files

If you want to keep track of what you eat, Pantry is flexible in where you store the foods you eat. You can choose to keep all the foods you ever eat in a single file. However, to make any sense of such a file, you may find that you have to enter `date` and `meal` traits for every food you enter. As we've seen, that can require a lot of typing.

But remember, Pantry is flexible. You can instead decide to keep a separate file for each day, or a separate file for each meal. You can of course sort these into directories however you see fit. Remember that **pantry** is flexible and can search and print results from more than one file at a time.

If you do decide to keep a lot of foods in one file, the **pantry-addTo** program can help you, as we will discuss in a later section.

Chapter 4. More Pantry usage

Now that you have seen some practical examples of how to use Pantry, there will be even more information that you will find useful in this chapter.

4.1. Using the `--sort` option to sort foods within a report

As you have seen before, Pantry does not sort foods when it stores them. This allows Pantry to store and retrieve foods very quickly, but it also means that your foods come out in a big jumble:

Example 4-1. Pantry does not sort foods

```
$ pantry --date "May 8" --print traits-blank diary
Carrots, raw
Group: Vegetables and Vegetable Products
Date: May 8 Meal: Lunch
Refuse: 11 percent Crown, tops and scrapings
1 large (7-1/4" to 8-/1/2" long) (72g)

Cereals ready-to-eat, KELLOGG, KELLOGG'S Corn Flakes
Group: Breakfast Cereals
Date: May 8 Meal: Breakfast
1 cup (1 NLEA serving) (28g)

Milk, reduced fat, fluid, 2% milkfat, with added vitamin A
Group: Dairy and Egg Products
Date: May 8 Meal: Breakfast
1 cup (244g)

McDONALD'S, French Fries
Group: Fast Foods
Date: May 8 Meal: Dinner
1 large serving (170g)

Ice creams, chocolate, rich
Group: Sweets
Date: May 8 Meal: Lunch
1 cup (148g)

Chicken, broilers or fryers, wing, meat and skin, cooked, roasted
Group: Poultry Products
Date: May 8 Meal: Lunch
Refuse: 48 percent Bone
3 wing, bone removed (102g)

Apples, raw, with skin
Group: Fruits and Fruit Juices
Date: May 8 Meal: Breakfast
```

```
Refuse: 8 percent Core and stem
1 large (3-1/4" dia) (approx 2 per lb) (212g)

Snacks, popcorn, oil-popped, unsalted
Group: Snacks
Date: May 8 Meal: Dinner
2 oz (57g)

McDONALD'S, Cheeseburger
Group: Fast Foods
Date: May 8 Meal: Dinner
1 item (119g)
```

This is of course a bit hard to comprehend. Of course, I wouldn't be bringing this up if there weren't a solution at hand: the `--sort` option. Use it and Pantry will sort the buffer before it is printed. The `--sort` option takes a single argument to indicate how you want your foods sorted. This argument consists of a series of letters, with each letter being the first letter of the trait you wish to use as a sorting key. For example, to sort by `meal` and then by `name`, use `--sort mn`. Lower-case letters sort in ascending order, while upper-case letters sort in descending order. So, to sort the meal names in ascending order and then the food names in descending order, use `--sort mN`:

Example 4-2. Using the `--sort` option

```
$ pantry --date "May 8" --sort mN --print traits-blank diary
Milk, reduced fat, fluid, 2% milkfat, with added vitamin A
Group: Dairy and Egg Products
Date: May 8 Meal: Breakfast
1 cup (244g)

Cereals ready-to-eat, KELLOGG, KELLOGG'S Corn Flakes
Group: Breakfast Cereals
Date: May 8 Meal: Breakfast
1 cup (1 NLEA serving) (28g)

Apples, raw, with skin
Group: Fruits and Fruit Juices
Date: May 8 Meal: Breakfast
Refuse: 8 percent Core and stem
1 large (3-1/4" dia) (approx 2 per lb) (212g)

Snacks, popcorn, oil-popped, unsalted
Group: Snacks
Date: May 8 Meal: Dinner
2 oz (57g)

McDONALD'S, French Fries
Group: Fast Foods
Date: May 8 Meal: Dinner
1 large serving (170g)

McDONALD'S, Cheeseburger
```

```
Group: Fast Foods
Date: May 8 Meal: Dinner
1 item (119g)

Ice creams, chocolate, rich
Group: Sweets
Date: May 8 Meal: Lunch
1 cup (148g)

Chicken, broilers or fryers, wing, meat and skin, cooked, roasted
Group: Poultry Products
Date: May 8 Meal: Lunch
Refuse: 48 percent Bone
3 wing, bone removed (102g)

Carrots, raw
Group: Vegetables and Vegetable Products
Date: May 8 Meal: Lunch
Refuse: 11 percent Crown, tops and scrapings
1 large (7-1/4" to 8-/1/2" long) (72g)
```

Remember that all traits in Pantry are strings, including the `date` `pctRefuse`, and `qty` traits. These traits are sorted as strings rather than as numbers or as dates. This can lead to unexpected results. For example, sorting the strings `1`, `2`, and `10` will yield `1`, `10`, `2`.

Similarly, when sorting dates, `7-1`, `7-10`, and `7-2` will not yield the results you may expect. An easy fix for this is to use leading zeroes; sorting `07-01`, `07-02`, and `07-10` will do what you expect.

As you can see above, the `meal` traits are sorted in alphabetical order, yielding `Breakfast`, `Dinner`, and `Lunch`. You can, however, sort these into whatever order you wish, such as the more logical `Breakfast`, `Lunch`, and `Dinner`. We'll find out how to do this when we talk about configuring Pantry, later.

Finally, perhaps you want to be able to sort your foods into any arbitrary order. The `order` trait is useful for this purpose. To sort foods into any order, just assign appropriate `order` values for each food (perhaps *`a`*, *`b`*, *`c`* or *`010`*, *`020`*, and *`030`*) and then use `--sort o`. Pantry can even help you by assigning values to the `order` trait automatically, as we will see in the next section.

4.2. More change options

4.2.1. Using the `--auto-order` option

Often you will find that you enter foods into Pantry in the same order in which you eat them. You might find it handy to have Pantry show those foods to you in the same order in which you entered them. The

`--auto-order` option is handy for this.

When you add foods to a file and you have chosen the `--auto-order`, Pantry will automatically change the `order` trait of the added foods. To find what the new `order` trait should be, Pantry first searches the file that the food is being added to for other foods with values for the `date` and `meal` traits that are identical to the ones in the food that is being added. Pantry then sorts these foods in ascending order by their `order` trait.

Pantry then checks to see if the highest food's `order` trait matches the regular expression `^[0-9]{4}$`. Examples of values for which this would be true include *`0010`*, *`2338`*, and *`0100`*. Pantry then takes that value, removes any leading zeroes, removes the last digit, and increments the result by one. A trailing zero is added to the result, and the result is left-padded with zeroes if necessary in order to make it four digits. Thus, if the previously highest `order` trait was *`0010`*, the resulting `order` is *`0020`*. *`2338`* yields a new order of *`2340`*, and *`0100`* leads to *`0110`*.

If the highest food's `order` trait does *not* match the regular expression `^[0-9]{4}$`, then the new food's `order` trait is set to *`0010`*.

As always, an example helps. We'll use the same foods as we did in a previous example, but let's say you ate the exact same thing one day later:

Example 4-3. Using --auto-order

```
$ pantry --ignore-case --name "apples, raw, with skin" --c-qty \
> 1 --c-unit large --c-date "May 9" --c-meal Breakfast --auto- \
> order --add diary master
$ pantry --ignore-case --name "kellogg's corn flakes" \
> --c-qty 1 --c-unit cup --c-date "May 9" --c-meal Breakfast \
> --auto-order --add diary master
$ pantry --ignore-case --name "milk, reduced fat, fluid, 2% \
> milkfat, with added vitamin A" --c-qty 1 --c-unit cup --c-date \
> "May 9" --c-meal Breakfast --auto-order --add diary master
$ pantry --ignore-case --name "carrots, raw" --c-qty 1 \
> --c-unit ^large --c-date "May 9" --c-meal Lunch --auto-order \
> --add diary master
$ pantry --ignore-case --name "Chicken, broilers or fryers, wing, \
> meat and skin, cooked, roasted" --c-date "May 9" \
> --c-unit wing --c-qty 3 --c-meal Lunch --auto-order --add diary master
$ pantry --ignore-case --name "ice creams, chocolate, rich" \
> --c-qty .5 --c-unit cup --c-date "May 9" --c-meal \
> "Lunch" --auto-order --add diary master
$ pantry --ignore-case --name "mcdonald's, cheeseburger" \
> --c-unit item --c-qty 1 --c-date "May 9" --c-meal \
> "Dinner" --auto-order --add diary master
$ pantry --ignore-case --name "mcdonald's, french fries" \
> --c-qty 1 --c-unit large --c-date "May 9" --c-meal Dinner \
> --auto-order --add diary master
$ pantry --ignore-case --name "popcorn, oil-popped, unsalted" \
> --c-qty '2 1/2' --c-unit oz --c-date "May 9" \
```

```
> --c-meal Dinner --auto-order --add diary master
```

Now you can sort the foods by their `order` traits:

Example 4-4. Using --sort with foods that have been automatically ordered

```
$ pantry --date "May 9" --print traits-blank --sort dmo diary
Apples, raw, with skin
Group: Fruits and Fruit Juices
Date: May 9 Meal: Breakfast Order: 0010
Refuse: 8 percent Core and stem
1 large (3-1/4" dia) (approx 2 per lb) (212g)

Cereals ready-to-eat, KELLOGG, KELLOGG'S Corn Flakes
Group: Breakfast Cereals
Date: May 9 Meal: Breakfast Order: 0020
1 cup (1 NLEA serving) (28g)

Milk, reduced fat, fluid, 2% milkfat, with added vitamin A
Group: Dairy and Egg Products
Date: May 9 Meal: Breakfast Order: 0030
1 cup (244g)

McDONALD'S, Cheeseburger
Group: Fast Foods
Date: May 9 Meal: Dinner Order: 0010
1 item (119g)

McDONALD'S, French Fries
Group: Fast Foods
Date: May 9 Meal: Dinner Order: 0020
1 large serving (170g)

Snacks, popcorn, oil-popped, unsalted
Group: Snacks
Date: May 9 Meal: Dinner Order: 0030
2 1/2 oz (71g)

Carrots, raw
Group: Vegetables and Vegetable Products
Date: May 9 Meal: Lunch Order: 0010
Refuse: 11 percent Crown, tops and scrapings
1 large (7-1/4" to 8-/1/2" long) (72g)

Chicken, broilers or fryers, wing, meat and skin, cooked, roasted
Group: Poultry Products
Date: May 9 Meal: Lunch Order: 0020
Refuse: 48 percent Bone
3 wing, bone removed (102g)

Ice creams, chocolate, rich
Group: Sweets
```

```
Date: May 9 Meal: Lunch Order: 0030
.5 cup (74g)
```

The results are a bit different if we set only the `date` trait when we use `--auto-order`:

Example 4-5. Using `--auto-order` with only the `date` trait changed

```
$ pantry --ignore-case --name "apples, raw, with skin" --c-qty \
> 1 --c-unit large --c-date "May 10" --auto-order --add diary \
> master
$ pantry --ignore-case --name "kellogg's corn flakes" \
> --c-qty 1 --c-unit cup --c-date "May 10" --auto-order --add \
> diary master
$ pantry --ignore-case --name "milk, reduced fat, fluid, 2% \
> milkfat, with added vitamin A" --c-qty 1 --c-unit cup --c-date \
> "May 10" --auto-order --add diary master
$ pantry --ignore-case --name "carrots, raw" --c-qty 1 \
> --c-unit ^large --c-date "May 10" --auto-order --add diary \
> master
$ pantry --ignore-case --name "Chicken, broilers or fryers, wing, \
> meat and skin, cooked, roasted" --c-date "May 10" \
> --c-unit wing --c-qty 3 --auto-order --add diary master
$ pantry --ignore-case --name "ice creams, chocolate, rich" \
> --c-qty 1 --c-unit cup --c-date "May 10" --auto-order --add \
> diary master
$ pantry --ignore-case --name "mcdonald's, cheeseburger" \
> --c-unit item --c-qty 1 --c-date "May 10" --auto-order --add \
> diary master
$ pantry --ignore-case --name "mcdonald's, french fries" \
> --c-qty 1 --c-unit large --c-date "May 10" --auto-order \
> --add diary master
$ pantry --ignore-case --name "popcorn, oil-popped, unsalted" \
> --c-qty 2 --c-unit oz --c-date "May 10" --auto-order --add \
> diary master
$ pantry --date "May 10" --print traits-blank --sort o diary
Apples, raw, with skin
Group: Fruits and Fruit Juices
Date: May 10 Order: 0010
Refuse: 8 percent Core and stem
1 large (3-1/4" dia) (approx 2 per lb) (212g)

Cereals ready-to-eat, KELLOGG, KELLOGG'S Corn Flakes
Group: Breakfast Cereals
Date: May 10 Order: 0020
1 cup (1 NLEA serving) (28g)

Milk, reduced fat, fluid, 2% milkfat, with added vitamin A
Group: Dairy and Egg Products
Date: May 10 Order: 0030
1 cup (244g)
```

```
Carrots, raw
Group: Vegetables and Vegetable Products
Date: May 10 Order: 0040
Refuse: 11 percent Crown, tops and scrapings
1 large (7-1/4" to 8-/1/2" long) (72g)

Chicken, broilers or fryers, wing, meat and skin, cooked, roasted
Group: Poultry Products
Date: May 10 Order: 0050
Refuse: 48 percent Bone
3 wing, bone removed (102g)

Ice creams, chocolate, rich
Group: Sweets
Date: May 10 Order: 0060
1 cup (148g)

McDONALD'S, Cheeseburger
Group: Fast Foods
Date: May 10 Order: 0070
1 item (119g)

McDONALD'S, French Fries
Group: Fast Foods
Date: May 10 Order: 0080
1 large serving (170g)

Snacks, popcorn, oil-popped, unsalted
Group: Snacks
Date: May 10 Order: 0090
2 oz (57g)
```

You might complain that the previous examples took a lot of typing. We'll learn how you can cut down the amount of typing in a later section.

4.2.2. Changing food quantities by refuse amount

Internally, Pantry's nutrient amounts are recorded as amount of nutrient for a given amount of edible food. More specifically, Pantry tracks each of its foods by recording the amount of each nutrient per 100 grams of edible food. How Pantry converts from nutrient per 100 grams of edible food to the total amount of nutrient in your foods varies depending upon the unit of your food.

The units `g`, `oz`, and `lb` are available for every food. For foods using one of these available units, it is simple for Pantry to convert from nutrients per 100 grams to nutrients per ounce or nutrients per pound.

Any of a food's other avaialble units are recorded in Pantry with their total grams of edible portion. Pantry then uses this gram weight and the amount of nutrient per 100 grams of edible portion to compute the total nutrients in your food.

An example will prove helpful. Let' take some apples:

Example 4-6. Traits and available units for an apple

```
$ pantry --name "Apples, raw, with skin" --print names-units \
> master
Apples, raw, with skin
   cup, quartered or chopped
   large (3-1/4" dia) (approx 2 per lb)
   NLEA serving
   medium (2-3/4" dia) (approx 3 per lb)
   cup slices
   small (2-1/2" dia) (approx 4 per lb)
```

By now this looks familiar to you. This way, if you eat a large apple, you change the unit of the food to *`large (3-1/4" dia) (approx 2 per lb)`*. Pantry knows how many grams a large apple weighs. To find out how many grams correspond to each unit, use the *`measures`* report, which shows the weight in grams of each measure:

Example 4-7. Using the *`measures`* report

```
$ pantry --name "Apples, raw, with skin" --print names- \
> measures master
Apples, raw, with skin
   cup, quartered or chopped (125g)
   large (3-1/4" dia) (approx 2 per lb) (212g)
   NLEA serving (154g)
   medium (2-3/4" dia) (approx 3 per lb) (138g)
   cup slices (110g)
   small (2-1/2" dia) (approx 4 per lb) (106g)
```

The weights in grams correspond to the *edible* portion of the food. Thus, in this example, a typical large apple weighs 212 grams, without the inedible portions (such as the core).

But perhaps you eat an apple and you want to use the *`g`*, *`oz`*, or *`lb`* unit. This is especially likely if, for example, you have a kitchen scale and you wish to precisely measure the amount of all the food you eat. If you wish to weigh your own apples rather than using the approximations given in the available units, you have a few options.

One option is to simply cut up the apple, cut out the core, and then weigh the resulting chunks. That is fine if you usually cut up your apples (as I do) but not so good if you do not want to chop up your apples all the time.

Another option is to weigh the entire apple, whole, and record that you ate (for example) a seven-ounce apple. The only problem with this is that the nutrient amounts in Pantry are for the edible portion of the food. Thus, if you do not account for the fact that a fraction of the apple is a core and a stem, you will overestimate how much apple you ate.[1] To compensate for this, you could weigh the entire apple, eat it, and then weigh the leftover core and stem. You would subtract the weight of the core and stem from the weight of the entire apple, and then figure that you ate the difference. That would get tedious.

A third option is to use the `--refuse` option. It uses the `refuse` trait to reduce your food's quantity by the percentage of refuse. As you saw in the example above, an apple is 8 percent refuse. So, assuming you have an eight-ounce apple:[2]

Example 4-8. Using the `--refuse` option

```
$ pantry --name "Apples, raw, with skin" --c-qty 8 --c-unit oz \
> --print traits-nuts --refuse master
Apples, raw, with skin
Group: Fruits and Fruit Juices
Refuse: 8 percent Core and stem
7.36 oz (209g)
Nutrient                  Amount          %G     %TOT
-----------------------------------------------------
Calories                  109  kcal        5      100
Total Fat                 0    g           1      100
Saturated Fat             0    g           0      100
Cholesterol               0    mg          0      NA
Sodium                    2    mg          0      100
Total Carbohydrate        29   g           10     100
Dietary Fiber             5    g           20     100
Sugars                    22   g           NG     100
Protein                   1    g           1      100
Vitamin A                 113  IU          2      100
Vitamin C                 10   mg          16     100
Calcium                   13   mg          1      100
Iron                      0    mg          1      100
```

As you can see, the `--refuse` option reduces the quantity of the apple by eight percent. This way, you can weigh the entire apple (including the core) and still get a good estimate of which portion of the apple was actually edible.

It only makes sense to use the `--refuse` option when you are using the *`g`*, *`oz`*, or *`lb`* units. Every other unit in the `master` file already accounts for refuse--for example, with the apple, the *`large (3-1/4" dia) (approx 2 per lb)`* already accounts for the eight percent refuse.

When using the `--refuse` option, every food in the buffer will have its quantity reduced by its corresponding refuse percentage. This of course does not affect foods with zero percent refuse. If a food's refuse percentage is set to *`None`*, **pantry** prints a warning message but does not adjust the quantity of the food.

4.3. Configuring Pantry

The Pantry configuration file is XML. If you know nothing about XML, the Wikipedia article (http://en.wikipedia.org/wiki/XML) is a good place to start.

In this section we'll walk through how to create your own Pantry configuration file. The default name for this file is `~/.pantryrc.xml`. If you want to use a different filename for your configuration file, you may specify it by setting an environment variable named `PANTRYRC`.

So, to get started open a text file named `~/.pantryrc.xml` in your favorite text editor. Every `.pantryrc.xml` file contains the root element `pantry-config`, so start by including that in your `.pantryrc.xml`:

Example 4-9. Beginning a `.pantryrc.xml` file

```
<pantryrc>
</pantryrc>
```

4.3.1. Configuring a search path

You can have **pantry** search several directories for files that you are searching. The concept is similar to the `PATH` variable that shells use. This way, you need not be in the same directory as the files you are searching, and you will not need to specify the entire path.

The search path feature works only for files that you are searching for foods. It has no effect on the `--add` option, for example; the `--add` option will always assume its argument is in the current directory, unless you specify a pathname in the argument. The search path works similarly to the same feature in shells: the path is searched only if the filename does not include a slash. If the filename does include a slash, then only the appropriate directory is searched for the file.

If you do not specify a search path, Pantry will search only the current directory for files.

To specify a search path, create one `path` element for each directory you wish to be in your path. Each element has a single attribute, `dir`. As with shells, you must specify the current directory if you want **pantry** to include it when searching. To specify the current directory, use a single period.

The next example specifies two directories, the current directory and the `/home/massysett/pantry` directory, for the search path.

Example 4-10. `.pantryrc.xml` with search path specified

```
<pantryrc>
    <path dir='.' />
```

```
    <path dir='/home/massysett/pantry' />
</pantryrc>
```

4.3.2. Specifying your own nutrient lists, and a default nutrient list

Using your `.pantryrc.xml`, you may specify your own nutrient lists. You may also specify which nutrient list is the default--that is, which nutrient list Pantry uses when you do not use the `--nutrient-list` option.

4.3.2.1. Specifying a default nutrient list

The default nutrient list is the one Pantry uses when you do not specify a nutrient list using the `--nutrient-list` option. The default nutrient list can be one of the nutrient lists included with Pantry, or it can be a nutrient list you create yourself. You can set a default nutrient list even if you do not define any of your own nutrient lists, but of course then you will need to choose only from one of the nutrient lists included with Pantry. If you do not specify a default nutrient list, Pantry will use *`facts`*.

To specify a default nutrient list, add a `default-nutrient-list` element. This element has a single attribute, `name`, which specifies the name of the default nutrient list. This name is case-sensitive. In the following example, the default nutrient list is *`facts`*.

Example 4-11. Specifying a default nutrient list in `.pantryrc.xml`

```
<pantryrc>
    <path dir='.' />
    <path dir='/home/massysett/pantry' />

    <default-nutrient-list name='facts' />
</pantryrc>
```

4.3.2.2. Specifying your own nutrient lists

You may specify nutrient lists of your own in your `.pantryrc.xml` file. Each nutrient list you create will reside inside a `nutrient-list` element. To define multiple nutrient lists, you may define multiple `nutrient-list` elements; each `nutrient-list` element must be the direct descendant of the `nutrient-lists` element.

Each `nutrient-list` element has a single attribute, `name`, which defines the name of the nutrient list. `name` must begin with a letter, with subsequent characters being either letters or numbers. The `nutrient-list` element is parent to one or more empty `nutrient` elements.

Each `nutrient` element has three attributes. The first is `name`, such as *`Calories`* or *`Protein`*. The second is `units`, or the units corresponding to the `name`. For *`Calories`* this would be *`kcal`*; for *`Protein`*, this would be *`g`*. The third attribute, `goal`, is optional. If you wish to set a goal for your intake of this nutrient, set attribute equal to your numeric goal. If on the other hand you are including this nutrient in your nutrient list only because you want to see this nutrient printed in reports, you may set the `goal` attribute to the empty string (`""`) or you may simply omit this attribute.

The `name` and `units` values you use in your `nutrient` elements must correspond to one of the possible nutrients in Pantry. To see a list of all the possible nutrients in Pantry, use `--nutrient-list all --print list`:

Example 4-12. Showing all available nutrients

```
$ pantry --nutrient-list all --print list master
Nutrient Name             Units          Goal
-------------------------------------------------
10:0                      g              None
12:0                      g              None
13:0                      g              None
14:0                      g              None
14:1                      g              None
15:0                      g              None
15:1                      g              None
16:0                      g              None
16:1 c                    g              None
16:1 t                    g              None
16:1 undifferentiated     g              None
17:0                      g              None
17:1                      g              None
[ trimmed to save space ]
```

4.3.2.2.1. Specifying your own nutrient lists: an example

To get started, let's suppoose I made a New Year's resolution to eat less. I want to eat 1800 calories a day. I also want to make sure I eat 40 grams of fat per day, 90 grams of protein a day, and 270 grams of carbs a day. I'm also curious about how much calcium I am taking in, but I don't want to bother setting a goal for that. I decide to create a new nutrient list called *`eatless`* to help me meet my goals. Here is what my `.pantryrc.xml` would look like:

Example 4-13. `.pantryrc.xml` with new nutrient list

```
<pantryrc>
    <path dir='.' />
    <path dir='/home/massysett/pantry' />

    <default-nutrient-list name='facts' />
    <nutrient-list name='eatless'>
        <nutrient name='Calories' units='kcal' goal='1800' />
        <nutrient name='Total Fat' units='g' goal='40' />
```

```
        <nutrient name='Protein' units='g' goal='90' />
        <nutrient name='Total Carbohydrate' units='g' goal='270' />
        <nutrient name='Calcium' units='mg' />
    </nutrient-list>
</pantryrc>
```

To use my new nutrient list, I can just specify `--nutrient-list eatless` when I use **pantry**:

Example 4-14. Using a new nutrient list

```
$ pantry --name "Apples, raw, with skin" --print traits-nuts \
> --nutrient-list eatless master
Apples, raw, with skin
Group: Fruits and Fruit Juices
Refuse: 8 percent Core and stem
100 g (100g)
Nutrient                  Amount         %G     %TOT
-----------------------------------------------------
Calories                  52   kcal       3      100
Total Fat                 0    g          0      100
Protein                   0    g          0      100
Total Carbohydrate        14   g          5      100
Calcium                   6    mg         NG     100
```

If I wanted to make *eatless* the new default nutrient list so that **pantry** would use it even if I do not use the `--nutrient-list eatless`, I would add **<default-list name='eatless' />** right after any `path` elements in my `.pantryrc.xml`.

4.3.3. Specifying custom sort orders

As we learned previously, you can use the `--sort` to sort the reports that Pantry prints by the traits of the foods. But this only sorts the strings into your system's collation order. What if you want to sort something into an arbitrary order? For instance, you might want to sort your meals into the order `Breakfast`, `Lunch`, and `Dinner`, which certainly is not alphabetical order.

To do this you can specify a sort order in your `antryrc.xml` file. You may do this for any trait that is not a numeric trait--that is, for any trait except the `qty` and `pctRefuse` traits. To do this, include an element named `sort-order` in your `.pantryrc.xml` file. This element must have a single attribute, `trait`. For example, here we wish to specify a custom sort order for the `meal` trait, so we set the `trait` attribute to *meal*.

Inside the `sort-order` element, include a single `item` element for each value you wish to include in your sort order. Each `item` element must have a single attribute, `value`. Here, for example, is how to sort

meals into the order `Breakfast`, `Lunch`, and `Dinner`. If the value of a food's trait is not in the sort order list (for example, *`supper`*, it is sorted alphabetically with other foods whose trait value is not in the list.

Example 4-15. `.pantryrc.xml` with `sort-order` element

```
<pantryrc>
    <path dir='.' />
    <path dir='/home/massysett/pantry' />

    <default-nutrient-list name='facts' />
    <nutrient-list name='eatless'>
        <nutrient name='Calories' units='kcal' goal='1800' />
        <nutrient name='Total Fat' units='g' goal='40' />
        <nutrient name='Protein' units='g' goal='90' />
        <nutrient name='Total Carbohydrate' units='g' goal='270' />
        <nutrient name='Calcium' units='mg' />
    </nutrient-list>

    <sort-order trait='meal'>
        <item value='Breakfast' />
        <item value='Lunch' />
        <item value='Dinner' />
    </sort-order>
</pantryrc>
```

4.4. Changing food quantities by nutrient amount

pantry features a `--by-nut` change option that will automatically change the quantity of a food so that it has a particular amount of a certain nutrient that you specify.[3] The `--by-nut` option takes two arguments. The first argument is a regular expression to match the nutrient you wish to use. The second argument is the amount of that nutrient that you wish the food to be set to. Let us look at an example:

Example 4-16. Using the `--by-nut` option

```
$ pantry --by-nut Calories 200 --name "Apples, raw, with skin" \
> --print traits-nuts master
Apples, raw, with skin
Group: Fruits and Fruit Juices
Refuse: 8 percent Core and stem
384.62 g (385g)
Nutrient                  Amount         %G     %TOT
-------------------------------------------------------
Calories                  200  kcal       10     100
Total Fat                 1    g          1      100
Saturated Fat             0    g          1      100
Cholesterol               0    mg         0      NA
Sodium                    4    mg         0      100
Total Carbohydrate        53   g          18     100
```

```
Dietary Fiber             9    g          37     100
Sugars                    40   g          NG     100
Protein                   1    g          2      100
Vitamin A                 208  IU         4      100
Vitamin C                 18   mg         29     100
Calcium                   23   mg         2      100
Iron                      0    mg         3      100
```

As you can see, Pantry automatically changed the quantity of apples so that you would have 200 calories of apple. This works with any nutrient:

Example 4-17. Using `--by-nut` with Total Fat

```
$ pantry --by-nut "Total Fat" 5 --name "Avocados, raw, \
> California" --print traits-nuts master
Avocados, raw, California
Group: Fruits and Fruit Juices
Refuse: 33 percent Seed and skin
32.45 g (32g)
Nutrient                  Amount         %G     %TOT
-----------------------------------------------------
Calories                  54   kcal       3      100
Total Fat                 5    g          8      100
Saturated Fat             1    g          3      100
Cholesterol               0    mg         0      NA
Sodium                    3    mg         0      100
Total Carbohydrate        3    g          1      100
Dietary Fiber             2    g          9      100
Sugars                    0    g          NG     100
Protein                   1    g          1      100
Vitamin A                 48   IU         1      100
Vitamin C                 3    mg         5      100
Calcium                   4    mg         0      100
Iron                      0    mg         1      100
```

That gave us the right number of grams of avocado that contain 5 grams of fat. The `--by-nut` also works with the `--c-unit`. For example, here is how you can find out what fraction of an avocado you would need to eat in order to consume 5 grams of fat:

Example 4-18. Using `--by-nut` with `--c-unit`

```
$ pantry --by-nut "Total Fat" 5 --name "Avocados, raw, \
> California" --c-unit fruit --print traits-nuts master
Avocados, raw, California
Group: Fruits and Fruit Juices
Refuse: 33 percent Seed and skin
0.24 fruit, without skin and seed (33g)
Nutrient                  Amount         %G     %TOT
-----------------------------------------------------
Calories                  55   kcal       3      100
Total Fat                 5    g          8      100
```

```
Saturated Fat             1    g          3      100
Cholesterol               0    mg         0      NA
Sodium                    3    mg         0      100
Total Carbohydrate        3    g          1      100
Dietary Fiber             2    g          9      100
Sugars                    0    g          NG     100
Protein                   1    g          1      100
Vitamin A                 48   IU         1      100
Vitamin C                 3    mg         5      100
Calcium                   4    mg         0      100
Iron                      0    mg         1      100
```

The `--by-nut` option is useful in at least two circumstances. First, as we have already seen, it is useful if you are wondering how much of a food you need to consume in order to get a certain amount of a given nutrient. Second, it is useful if you wish to approximate intake of a food that is not already in the `master` file. For example, suppose I eat 1/2 of a cup of Stonyfield After Dark Chocolate ice cream. (http://www.stonyfield.com/OurProducts/FrozenYogurtIceCream.cfm). Many brand-name foods are already in the `master` file, but this one is not. I look through the ice creams and find that the closest thing to Stonyfield in the `master` file is probably *`Ice creams, chocolate, rich`*:

Example 4-19. The closest thing to Stonyfield

```
$ pantry --exact-match --name "Ice creams, chocolate, rich" \
> --c-unit "cup" --c-qty .5 --print traits-nuts master
Ice creams, chocolate, rich
Group: Sweets
.5 cup (74g)
Nutrient                  Amount         %G     %TOT
-----------------------------------------------------
Calories                  189  kcal       9      100
Total Fat                 13   g          19     100
Saturated Fat             8    g          38     100
Cholesterol               44   mg         15     100
Sodium                    42   mg         2      100
Total Carbohydrate        15   g          5      100
Dietary Fiber             1    g          3      100
Sugars                    13   g          NG     100
Protein                   3    g          7      100
Vitamin A                 528  IU         11     100
Vitamin C                 0    mg         1      100
Calcium                   105  mg         11     100
Iron                      1    mg         4      100
```

But Stonyfield apparently is even richer than this. 1/2 cup of *`Ice creams, chocolate, rich`* has 189 calories, but looking at the Stonyfield label tells me that 1/2 cup of Stonyfield has 250 calories. What should I do? Well, as we will learn in a later chapter, I can create a custom food for Stonyfield After Dark Chocolate ice cream. Or I might figure that *`Ice creams, chocolate, rich`* is close enough for our purposes. I can use it instead. My results will not be as accurate, but perhaps I am not feeling the need to be super accurate today. Because I know I ate 250 calories of Stonyfield, I just do this:

Example 4-20. Approximating one food by using another food and the `--by-nut` option

```
$ pantry --exact-match --name "Ice creams, chocolate, rich" \
> --by-nut Calories 250 --print traits-nuts master
Ice creams, chocolate, rich
Group: Sweets
98.04 g (98g)
Nutrient                  Amount         %G     %TOT
-----------------------------------------------------
Calories                  250  kcal       13     100
Total Fat                 17   g          26     100
Saturated Fat             10   g          51     100
Cholesterol               59   mg         20     100
Sodium                    56   mg         2      100
Total Carbohydrate        20   g          7      100
Dietary Fiber             1    g          4      100
Sugars                    17   g          NG     100
Protein                   5    g          9      100
Vitamin A                 699  IU         14     100
Vitamin C                 0    mg         1      100
Calcium                   139  mg         14     100
Iron                      1    mg         6      100
```

If you compare Pantry's output to the Stonyfield label (available at the website) you will see that Pantry comes out fairly close to what is on the label.

The first argument--the nutrient name--is a regular expression. That means that it is case-sensitive, like all other regular expressions in Pantry, unless you use the `--ignore-case` option. Thus, `--by-nut calories 250` will get you an error message unless you use `--ignore-case` but `--by-nut Calories 250` will always work. This also allows you to shorten things a bit--for example, you can use `--by-nut Saturated 250` to match Saturated Fat. In addition, the `--by-nut` also respects the `--exact-match` option, so if you are using `--exact-match`, then the first argument to `--by-nut` are not regular expressions and must exactly match the nutrient name.

Most commonly you would find yourself using `--by-nut` with Calories. Therefore, **pantry** has a `-K` option. It is equivalent to typing `--by-nut Calories`:

Example 4-21. Using the `-K` option

```
$ pantry --exact-match --name "Ice creams, chocolate, rich" -K \
> 250 --print traits-nuts master
Ice creams, chocolate, rich
Group: Sweets
98.04 g (98g)
Nutrient                  Amount         %G     %TOT
-----------------------------------------------------
Calories                  250  kcal       13     100
Total Fat                 17   g          26     100
Saturated Fat             10   g          51     100
Cholesterol               59   mg         20     100
```

```
Sodium                  56   mg          2      100
Total Carbohydrate      20   g           7      100
Dietary Fiber           1    g           4      100
Sugars                  17   g           NG     100
Protein                 5    g           9      100
Vitamin A               699  IU          14     100
Vitamin C               0    mg          1      100
Calcium                 139  mg          14     100
Iron                    1    mg          6      100
```

4.5. Using the pantry-addTo script

The **pantry-addTo** command is a front-end to the **pantry** command. If you use it you can type, for instance, **`ab banana`** to add a banana to today's breakfast.

pantry-addTo does this by examining the name of the file used to call it. In the above example, `ab` is a symlink to **pantry-addTo**. **pantry-addTo** "knows" that `ab` means "add to breakfast".

Here is how to configure **pantry-addTo**:

In your `.pantryrc.xml` file, add an element called `pantry-addTo`. Add one empty `symlink` element for each symlink you wish to use. The `symlink` element has two attributes. `filename` is the name of the symlink you will be using. It is only the basename of the symlink--that is, it is not the full pathname. The other attribute, `meal`, is the meal that will correspond to this symlink.

Add two more elements to your `.pantryrc.xml` file inside the `pantry-addTo` element. `source` specifies which file the foods will be copied from. It is a child to the `pantry-addTo` element and has one attribute, `filename`, which is the respective filename. You will probably find it easiest to use **pantry-addTo** if you specify a `source` that has only foods you use frequently, as we discussed earlier in Section 3.3.1.

You will also need a `destination` element, which specifies where new foods will be copied to. As with the `source` element, this element has a single attribute named `filename`.

In the following example, `ab` will add a food and change its `Meal` trait to *`Breakfast`*; `al` will add a food and change its `Meal` trait to *`Lunch`*, and so on. **pantry-addTo** will search the file `/home/massysett/pantry/quick` for foods, and the changed foods will be added to `/home/massysett/pantry/diary`.

Example 4-22. Example .pantryrc.xml file with pantry-addTo configuration

```
<pantryrc>
    <path dir='.' />
```

```
    <path dir='/home/massysett/pantry' />

    <default-nutrient-list name='facts' />
    <nutrient-list name='eatless'>
        <nutrient name='Calories' units='kcal' goal='1800' />
        <nutrient name='Total Fat' units='g' goal='40' />
        <nutrient name='Protein' units='g' goal='90' />
        <nutrient name='Total Carbohydrate' units='g' goal='270' />
        <nutrient name='Calcium' units='mg' />
    </nutrient-list>

    <sort-order trait='meal'>
        <item value='Breakfast' />
        <item value='Lunch' />
        <item value='Dinner' />
    </sort-order>

    <pantry-addTo>
        <symlink filename='ab' meal='Breakfast' />
        <symlink filename='al' meal='Lunch' />
        <symlink filename='ad' meal='Dinner' />
        <symlink filename='ab-dev' meal='Breakfast' />
        <symlink filename='al-dev' meal='Lunch' />
        <symlink filename='ad-dev' meal='Dinner' />

        <source filename='/home/massysett/pantry/quick' />
        <destination filename='/home/massysett/pantry/diary.txt' />
    </pantry-addTo>
</pantryrc>
```

Finally, create symlinks for each meal you specified in your .pantryrc. The link target is your `pantry-addTo` file, while the link name is the short name you specified in `pantry-addTo`. The symlinks will be easiest to use if they are in a directory that is in your PATH.

Using **pantry-addTo** is simple. Type the appropriate program name (such as **ab** in my example) followed by at least one, but no more than three, arguments. The first argument must be a regular expression that will search the `name` traits in the file you specified in `source`. The second argument, which is optional, is a quantity that you wish to change the quantity trait of your search results to. The third argument, also optional, is a unit argument, used to change the units of your search results if you wish. **pantry-addTo** will automatically print the traits of the foods it finds, after their traits have been changed.

Some further notes about using pantry-addTo:

- All searches are case insensitive.
- By default, **pantry-addTo** searches the `name` traits of the search file. Use the `-g` or `--group` option to search the `group` traits instead.
- There is an `-R` option that corresponds to the **pantry** option by the same name.
- **pantry-addTo** *always* uses the `--auto-order` option.

- **pantry-addTo** automatically sets the `date` trait of foods to today's date.
- **pantry-addTo** automatically prints the traits of the foods that it finds before adding them.

Here is an example of how to use **pantry-addTo**. First, it is best to use **pantry-addTo** with a `quick` file, so we will set one up first.

Example 4-23. Setting up a `quick` file

```
$ pantry --ignore-case --name "apples, raw, with skin" --c-qty \
> 1 --c-unit large --add quick master
$ pantry --ignore-case --name "kellogg's corn flakes" \
> --c-qty 1 --c-group "Cereal and Milk" --c-unit cup --add \
> quick master
$ pantry --ignore-case --name "milk, reduced fat, fluid, 2% \
> milkfat, with added vitamin A" --c-group "Cereal and \
> Milk" --c-qty 1 --c-unit cup --add quick master
$ pantry --ignore-case --name "carrots, raw" --c-qty 1 \
> --c-unit ^large --add quick master
$ pantry --ignore-case --name 'chicken.*wing, meat and skin, \
> cooked, roasted' --c-unit wing --c-qty 3 --add quick master
$ pantry --ignore-case --name "ice creams, chocolate, rich" \
> --c-qty 1 --c-unit cup --add quick master
$ pantry --ignore-case --name "mcdonald's, cheeseburger" \
> --c-unit item --c-qty 1 --add quick master
$ pantry --ignore-case --name "mcdonald's, french fries" \
> --c-qty 1 --c-unit large --add quick master
$ pantry --ignore-case --name "popcorn, oil-popped, unsalted" \
> --c-qty 2 --c-unit oz --add quick master
```

Now let's assume that you have set up your `.pantryrc.xml` and symlinks as we described above.

Example 4-24. Using pantry-addTo scripts

```
$ ab apples
Apples, raw, with skin
Group: Fruits and Fruit Juices
Date: 2007-07-20 Meal: Breakfast
Refuse: 8 percent Core and stem
1 large (3-1/4" dia) (approx 2 per lb) (212g)

$ ab -g "Cereal and Milk"
Cereals ready-to-eat, KELLOGG, KELLOGG'S Corn Flakes
Group: Cereal and Milk
Date: 2007-07-20 Meal: Breakfast
1 cup (1 NLEA serving) (28g)

Milk, reduced fat, fluid, 2% milkfat, with added vitamin A
Group: Cereal and Milk
Date: 2007-07-20 Meal: Breakfast
1 cup (244g)
```

```
$ al carrots 3 small
Carrots, raw
Group: Vegetables and Vegetable Products
Date: 2007-07-20 Meal: Lunch
Refuse: 11 percent Crown, tops and scrapings
3 small (5-1/2" long) (150g)

$ al -R chicken 10 '^oz$'
Chicken, broilers or fryers, wing, meat and skin, cooked, roasted
Group: Poultry Products
Date: 2007-07-20 Meal: Lunch
Refuse: 48 percent Bone
5.2 oz (147g)

$ ad cheeseburger 2
McDONALD'S, Cheeseburger
Group: Fast Foods
Date: 2007-07-20 Meal: Dinner
2 item (238g)

$ ad popcorn
Snacks, popcorn, oil-popped, unsalted
Group: Snacks
Date: 2007-07-20 Meal: Dinner
2 oz (57g)
```

Finally, let's see the results. Here we pair **pantry** with **date** to show only foods that match today's date in the `diary` file.

Example 4-25. Seeing the results from using pantry-addTo

```
$ pantry --date `date +%F` --sort mo --print traits-blank-sum diary
Apples, raw, with skin
Group: Fruits and Fruit Juices
Date: 2007-07-20 Meal: Breakfast Order: 0010
Refuse: 8 percent Core and stem
1 large (3-1/4" dia) (approx 2 per lb) (212g)

Cereals ready-to-eat, KELLOGG, KELLOGG'S Corn Flakes
Group: Cereal and Milk
Date: 2007-07-20 Meal: Breakfast Order: 0020
1 cup (1 NLEA serving) (28g)

Milk, reduced fat, fluid, 2% milkfat, with added vitamin A
Group: Cereal and Milk
Date: 2007-07-20 Meal: Breakfast Order: 0030
1 cup (244g)

McDONALD'S, Cheeseburger
Group: Fast Foods
Date: 2007-07-20 Meal: Dinner Order: 0010
```

```
2 item (238g)

Snacks, popcorn, oil-popped, unsalted
Group: Snacks
Date: 2007-07-20 Meal: Dinner Order: 0020
2 oz (57g)

Carrots, raw
Group: Vegetables and Vegetable Products
Date: 2007-07-20 Meal: Lunch Order: 0010
Refuse: 11 percent Crown, tops and scrapings
3 small (5-1/2" long) (150g)

Chicken, broilers or fryers, wing, meat and skin, cooked, roasted
Group: Poultry Products
Date: 2007-07-20 Meal: Lunch Order: 0020
Refuse: 48 percent Bone
5.2 oz (147g)

SUM:
Nutrient                  Amount         %G
-----------------------------------------------
Calories                  1744 kcal       87
Total Fat                 78   g          121
Saturated Fat             25   g          123
Cholesterol               227  mg         76
Sodium                    2021 mg         84
Total Carbohydrate        179  g          60
Dietary Fiber             18   g          73
Sugars                    60   g          NG
Protein                   87   g          175
Vitamin A                 27192 IU         544
Vitamin C                 27   mg         45
Calcium                   774  mg         77
Iron                      18   mg         100
```

4.6. Pantry usage shortcuts

4.6.1. Using short options

First, use the short options. For didactic purposes, all the examples in this manual use long-style options with two dashes. However, almost every **pantry** option is also available in short form. The search options (which search for a particular trait) and the change options (which change the traits of the foods in your result to a new value) have short options that conveniently differ only by their case. Thus, for example, `-g` searches by group name, and `-G` changes the group name.

If you need to jog your memory on what option does what, use the `--help` option.

In addition, when specifying reports, you may specify only the first few letters of the report name--enough letters to unambiguously specify the report name. For example, instead of typing **`--print names-nuts`**, you may instead type **`--print na-nu`**.

4.6.2. Learn your shell

One of the most powerful aspects of **pantry** is that you use it from your standard command shell. Shells have a wealth of features, such as history searching and command-line editing, that will speed up your shell usage. This will help you not only in Pantry but in other command-line tools as well. In addition, you can write scripts to automate your usage.

The most popular shell today is bash, which is very featureful. However, the bash documentation is dense and difficult to approach for a newcomer.

For a very well-documented shell that is easier to learn than bash, consider fish, the Friendly Interactive Shell (http://fishshell.org). For a very powerful shell with excellent documentation, try zsh (http://www.zsh.org/).

To improve your shell usage, I highly recommend getting a copy of From Bash to Z Shell: Conquering the Command Line (http://www.apress.com/book/bookDisplay.html?bID=361) by Kiddle, Peek, and Stephenson. It clearly explains many of the features of shells, such as command history and completion, that make interactive use much easier and faster. Unlike most texts on shells, this one focuses on interactive use. It is well worth the price.

4.6.3. Learn other Unix utilities

Because you use Pantry from your standard command shell, you can easily combine Pantry with other Unix utilities, in ways that even I cannot anticipate. Most obviously, you will often use Pantry with **less** so that you may easily scroll through reports. You might also find yourself using **sort**, **cut** or even **awk**. The possibilities are endless.

4.6.4. Use pantry-addTo

Finally, use the **pantry-addTo** command. It makes keeping track of daily food intake much, much easier.

4.7. Using Pantry with screen

Though Pantry may be used in conjunction with many other Unix shell utitlities, you may find GNU

Screen to be particularly useful. **screen** is so useful with Pantry due to its advanced cutting-and-pasting capabilities. Here is a good introduction (http://www.kuro5hin.org/story/2004/3/9/16838/14935) to **screen**. Most Linux systems will already have **screen** installed; if not, you'll be able to install it using your distribution's package manager.

Pantry includes the *paste* report, which makes **screen** particularly useful. Often in Pantry, you will be searching your `master` file, not entirely sure what food you are looking for. On other occasions, you will know what you are looking for, but your search terms return a lot of results to scroll through--and the food that you want has a very long name that you do not want to have to retype. Using the *paste* report, you can easily reuse your search results.

The *paste* report prints one line for each available unit for each food. The results include the `-x`, `-n`, and `-U` options, and the results are also quoted so you may easily paste them into a subsequent **pantry** command.

To see how powerful this is, we can return to our earlier example of finding how many calories are in an apple. You would likely start by searching the `master` file for the name trait *apple*. As we saw earlier, this returns 163 results:

Example 4-26. Searching for *apple* returns 163 results

```
$ pantry --ignore-case --name apple --print names master
Apple cider-flavored drink, powder, low calorie, with vitamin C, prepared
English muffins, raisin-cinnamon (includes apple-cinnamon)
Babyfood, cereal, high protein, with apple and orange, prepared with whole milk
Applesauce, canned, unsweetened, without added ascorbic acid
Pineapple, canned, light syrup pack, solids and liquids
Babyfood, fruit, bananas and pineapple with tapioca, strained
Babyfood, fruit, apple and raspberry,  junior
Apple juice, frozen concentrate, unsweetened, undiluted, with added ascorbic acid
Babyfood, apple-banana juice
Fruit salad, (peach and pear and apricot and pineapple and cherry), canned, extra heavy syrup, solids
Babyfood, fruit, apple and raspberry, strained
Pie fillings, apple, canned
Pineapple, frozen, chunks, sweetened
Scrapple, pork
Babyfood, fruit, bananas with apples and pears, strained
[ trimmed to save space ]
```

Earlier we limited our search results by typing subsequent **pantry** commands and tweaking the `--name` argument. When we zeroed in on the food we wanted, we typed yet another **pantry** command, this time to figure out the available units. Finally, we typed another **pantry** command to change the unit and quantity to what we wanted and to print the nutrient information for a single large apple.

You can do this using many fewer commands with **screen** and the *paste* report. After firing up **screen**, type a **pantry** for what you are looking for. I also use **sort** because I find it easier to scan results when they are alphabetized.

Example 4-27. Using the *paste* report

```
$ pantry --ignore-case --name apple --group fruit --print paste master | \
> sort
-x -n 'Apple juice, canned or bottled, unsweetened, with added ascorbic acid' -U 'cup'
-x -n 'Apple juice, canned or bottled, unsweetened, with added ascorbic acid' -U 'fl oz'
-x -n 'Apple juice, canned or bottled, unsweetened, with added ascorbic acid' -U 'g'
-x -n 'Apple juice, canned or bottled, unsweetened, with added ascorbic acid' -U 'lb'
-x -n 'Apple juice, canned or bottled, unsweetened, with added ascorbic acid' -U 'oz'
-x -n 'Apple juice, canned or bottled, unsweetened, without added ascorbic acid' -U 'cup'
-x -n 'Apple juice, canned or bottled, unsweetened, without added ascorbic acid' -U 'drink box (8.45 
-x -n 'Apple juice, canned or bottled, unsweetened, without added ascorbic acid' -U 'fl oz'
-x -n 'Apple juice, canned or bottled, unsweetened, without added ascorbic acid' -U 'g'
-x -n 'Apple juice, canned or bottled, unsweetened, without added ascorbic acid' -U 'lb'
-x -n 'Apple juice, canned or bottled, unsweetened, without added ascorbic acid' -U 'oz'
-x -n 'Apple juice, frozen concentrate, unsweetened, diluted with 3 volume water, with added ascorbic
-x -n 'Apple juice, frozen concentrate, unsweetened, diluted with 3 volume water, with added ascorbic
-x -n 'Apple juice, frozen concentrate, unsweetened, diluted with 3 volume water, with added ascorbic
-x -n 'Apple juice, frozen concentrate, unsweetened, diluted with 3 volume water, with added ascorbic
[ trimmed to save space ]
```

Now you can scroll through the results using Screen's commands; copy the line you are interested in to Screen's paste buffer. For more details on how to do this, consult the Screen manual page under "copy". Pay particular attention to the J command, which joins lines--this is useful if a particular line is so long that it does not fit on one line in your terminal.[4]

After finding the result you are interested in, you can easily paste it in to a new command line and add the options you need.

Notes

1. To which you might say: "big deal! It's just an apple core." The problem is minor with an apple, but consider the same issue with an ear of corn on the cob, or a pork sparerib...
2. Nowadays I frequently see single apples that weigh nearly a pound!
3. This feature shamelessly copied from a similar feature in NUT, as described under "Foods That Are Not in the Database" on How I Use NUT (http://www.lafn.org/~av832/usenut.html).
4. You don't have to use **screen**; you can copy and paste using your terminal emulator if you wish. Though I think **screen** is easier to use, you might diasgree...

Chapter 5. Adding new foods and editing the nutrients or units of existing foods

The `master` file comes with over 7,000 foods. However, you might find that you also need to add additional foods to Pantry. This chapter will tell you how to add additional foods by creating an XML file.

If you have a prepackaged food that already has a nutrition label, you can easily add it to a file. If however you have multiple foods that you wish to combine, then what you want is a recipe. We will discuss recipes in the next chapter.

As you may recall from our earlier discussions, an essential element of Pantry is the food. Foods are contained in food files. There are several different kinds of food files. Until now, all the food files we have used have been Pantry native files. You have been able to copy foods from one Pantry native file, such as the `master` file, and change the traits of those foods. To create entirely new foods, you will create a different type of food file, which we will call a *Foods XML* file. Unlike a Pantry native file, the Foods XML file is in human-readable, plain-text XML, which allows you to create new foods.

5.1. Creating a new food

5.1.1. Create a new blank XML file

In your favorite text editor, create a new file whose name ends with `.xml`. As an example, we will create a file named `foods.xml`.

The root element of a Foods XML file is a `foods` element, so to get started create that element.

Example 5-1. `foods.xml` file with `foods` element

```
<foods>
</foods>
```

5.1.2. Create a `food` element

Each food you create in the `foods.xml` file is contained within a `food` element. The `food` element has several attributes. Each corresponds to the various food traits that we've discussed earlier: `name`, `group`, `date`, `meal`, `unit`, `qty`, and `comment`. Only the `unit` and `qty` attributes are required. The `unit` attribute may contain *`g`*, *`oz`*, or *`lb`*, or it may contain one of the units given in the `units` element, which we will discuss shortly. The `qty` attribute must contain either an integer or a floating-point number.

For the other attributes, which are optional, **pantry** will create a zero-length attribute for those you leave unset. Typically though, you will set the `name` and `group` attributes to something useful, while leaving many of the others blank.

Typically you will be getting your nutrition information from food labels. Usually these will indicate a serving size, as well as a serving in grams. You will set the `unit` and `qty` traits to match what is given on the food label. As an example, we will use a Clif Bar. You may find it helpful to follow along with its nutrition label, which you can find here (http://www.clifbar.com/eat/eat.cfm?location=bar&id=286).[1] Its label shows a serving size of 1 bar. So, we enter the following text into our `foods.xml` to start:

Example 5-2. `foods.xml` with `food` element

```
<foods>
    <food name="Clif Bar, Oatmeal Raisin Walnut" group="Snacks" refDesc="" pctRefuse="" uni
    </food>
</foods>
```

5.1.3. Create a `nutrients` element

Next, you will create nutrients for your food. You define each nutrient by creating a `nutrient` element. Each `nutrient` element has two required attributes and one optional attribute.

The `name` attribute to the `nutrient` element, which is required, gives the name of the nutrient, such as *`Calories`* or *`Iron`*. These names must correspond to one of the nutrients that is allowed in Pantry; to get a list of the allowed nutrients, run **`pantry --nutrient-list all --print list`**. The names are case sensitive, so although *`Saturated Fat`* is a valid nutrient, *`Saturated fat`* is not a valid nutrient.

The `amount` attribute to the `nutrient` element is also required, and it is a number (either floating-point or integer) indicating the amount of the corresponding nutrient.

The third attribute to the `nutrient` element, `units`, is optional. Setting this attribute to `%`, indicates that the `amount` attribute is a percentage of an FDA Daily Value. Pantry will convert the `amount` attribute to an appropriate amount. This is useful because vitamins and minerals are listed on food labels by percent of daily value, rather than by amount. If the `units` is any value other than `%`, Pantry ignores it.[2]

To continue with our Clif Bar example, here is the Clif Bar with its nutrient information. Because I am lazy, I did not enter all the nutrition information; instead, I just entered all the macronutrients and four of the vitamins and minerals.

Example 5-3. Food with `nutrients` element

```
<foods>
    <food name="Clif Bar, Oatmeal Raisin Walnut" group="Snacks" refDesc="" pctRefuse="" uni
        <nutrient name="Calories" amount="240"/>
```

```
        <nutrient name="Total Fat" amount="5"/>
        <nutrient name="Saturated Fat" amount="1"/>
        <nutrient name="Trans Fat" amount="0"/>
        <nutrient name="Cholesterol" amount="0"/>
        <nutrient name="Sodium" amount="130"/>
        <nutrient name="Potassium" amount="310" />
        <nutrient name="Total Carbohydrate" amount="43"/>
        <nutrient name="Dietary Fiber" amount="5"/>
        <nutrient name="Sugars" amount="20"/>
        <nutrient name="Protein" amount="10"/>
        <nutrient name="Vitamin A" units="%" amount="30"/>
        <nutrient name="Vitamin C" units="%" amount="100"/>
        <nutrient name="Calcium" units="%" amount="25"/>
        <nutrient name="Iron" units="%" amount="25"/>
        <unit name="bar" grams="68"/>
</food>
</foods>
```

5.1.4. Creating `unit` elements

Finally, it is time to enter units, which you do using the `unit` element. These elements map common units of measure to a food's weight in grams, and they correspond to to the available units that we discussed earlier. Thus, each `unit` element has two attributes. `name` corresponds to the name of the unit, such as *`cup`* or *`stick`* or *`box`*--whatever is appropriate. The `grams` attribute corresponds to whatever the weight of a single `name` is, in grams.

As always, Pantry will provide the units *`g`*, *`oz`*, and *`lb`* for you, so do not enter those in `unit` elements. However, if you entered anything other than *`g`*, *`oz`*, or *`lb`* for the `unit` attribute of the `food` element, you must define a corresponding `unit` element. Thus, because the `unit` attribute for our Clif Bar is `bar`, we must define a corresponding `unit` element. With that, we have a complete `foods.xml` file.

Example 5-4. Complete `foods.xml` file.

```
<foods>
    <food name="Clif Bar, Oatmeal Raisin Walnut" group="Snacks" refDesc="" pctRefuse="" uni
    <nutrient name="Calories" amount="240"/>
    <nutrient name="Total Fat" amount="5"/>
    <nutrient name="Saturated Fat" amount="1"/>
    <nutrient name="Trans Fat" amount="0"/>
    <nutrient name="Cholesterol" amount="0"/>
    <nutrient name="Sodium" amount="130"/>
    <nutrient name="Potassium" amount="310" />
    <nutrient name="Total Carbohydrate" amount="43"/>
    <nutrient name="Dietary Fiber" amount="5"/>
    <nutrient name="Sugars" amount="20"/>
    <nutrient name="Protein" amount="10"/>
    <nutrient name="Vitamin A" units="%" amount="30"/>
    <nutrient name="Vitamin C" units="%" amount="100"/>
```

```
    <nutrient name="Calcium" units="%" amount="25"/>
    <nutrient name="Iron" units="%" amount="25"/>
    <unit name="bar" grams="68"/>
</food>
</foods>
```

5.2. Using a Foods XML file

You can use Foods XML files, such as the `foods.xml` file we have created in this example, just as you would use a Pantry native file (the format used by the `master` file. For example:

Example 5-5. Use a Foods XML file as you would use a Pantry native file

```
$ pantry --name Clif --print traits-nuts foods.xml
Clif Bar, Oatmeal Raisin Walnut
Group: Snacks
1.0 bar (68g)
Nutrient                  Amount         %G     %TOT
-------------------------------------------------------
Calories                  240  kcal       12     100
Total Fat                 5    g          8      100
Saturated Fat             1    g          5      100
Cholesterol               0    mg         0      NA
Sodium                    130  mg         5      100
Total Carbohydrate        43   g          14     100
Dietary Fiber             5    g          20     100
Sugars                    20   g          NG     100
Protein                   10   g          20     100
Vitamin A                 1500 IU         30     100
Vitamin C                 60   mg         100    100
Calcium                   250  mg         25     100
Iron                      5    mg         25     100
```

However, I recommend adding the foods to a Pantry native file, because you will find it most convenient if the foods you create are mingled with the other foods you use, either in the `master` file or in a `quick` file, as we discussed earlier.[3] For example, to add the Clif Bar to my `master` file, I simply run **`pantry --add master foods.xml`**.

Pantry destroys data in Foods XML files!

You may insert comments into your XML file. In addition, you will probably format your XML in a certain way by using tabs and newlines. However, if you make changes to the XML file using `--add`, `--edit`, or `--delete`, Pantry will *destroy* this formatting and lose the comments. You will also find that, if you use `--add`, `--edit`, or `--delete`, Pantry will rearrange the order of the foods in your XML file and will even rearrange the order of the attributes.

For the time being,[4] one solution to this is to use only the `comment` attribute if you wish to comment on foods. Also, you can simply refrain from making changes to the file using Pantry; Pantry will not destroy data in XML files if it only reading data from them.

5.3. Using Foods XML files to edit units or nutrients of foods

As you already know, you can easily edit the traits of foods in Pantry. But what if you want to edit the nutrients or available units of a food? As you can probably guess now, the solution is to use a Foods XML file. Foods XML files behave just like Pantry native files. So if, for example, you wish to edit foods in the `master` file, you can simply create a Foods XML file with the foods you wish to edit. Pantry automatically creates a Foods XML file if you use the extension `.xml`. So, to create a Foods XML file with bananas, use **`pantry --add banana.xml --name 'Bananas, raw' master`**. You can then edit this food however you wish, and you can then add the results back to the `master` file using the `--add` option.

Remember that Pantry will not allow you to have two identical foods in a single file. Two foods are considered identical if their traits are identical, even if their nutrients or their available units differ. If foods are identical, Pantry will simply add *`(Copy x)`* to the food's `Comment` trait. Keep this in mind if, for example, you edit the banana in the `banana.xml` file and then add it back to the `master` file.

5.4. Validation of Foods XML files

Pantry automatically validates Foods XML files for you.[5] To see the DTD that Pantry uses for this purpose, run **`pantry --dump foodsDTD`**. You might, for instance, want to use this DTD to validate the XML yourself.

If any of your files do not validate, Pantry will print an error message along with the output of the validator. Study the output of the validator because it will usually help you fix the error.

The validator will often hang if the input file is very large. Therefore, Pantry will only validate files if they are below 300 kilobytes in size. It is unlikely that you will ever create XML files that large by hand, although you might have Pantry create an XML file that large (that's what you will get if you run, for example, **`pantry --add foods.xml master`**). Pantry will give you an error message if you try to use a file that large. One way to use them is to use the `--skip-valid`, which will turn off validation. This is of course a bad idea for files that you have edited by hand (because they might have errors that then will go unchecked) but skipping validation is fine for large files that were generated entirely by Pantry. Alternatively, you can use `--force-valid` to make Pantry validate all XML files, regardless of how large they are. This might cause Pantry to wait for an extremely long time for the validator to finish--the validator might even freeze if the file is extremely large.

Notes

1. Use of the Clif Bar in this example wholeheartedly implies my endorsement of Clif Bars. They are handy, tasty snacks.
2. Even though Pantry ignores this attribute if it is anything other than `%`, you might set it anyway. For instance, perhaps you want to use XSLT to print food labels. Your printout can easily incorporate the values of the `units` attributes if you wish.
3. In addition, Foods XML files are slower to process than Pantry native files. If a Foods XML file contains just a few foods, the difference is not noticeable. However, for more than several dozen foods, the speed difference might begin to add up. That is why Pantry does not use XML as its native file format.
4. This problem likely will be fixed in a future release, but because of the intended use of Foods XML files as being primarily a way to import new foods, fixing this issue is a low priority.
5. If your system has **xmllint** installed, that is what Pantry will use to validate files. If not, then Pantry will use a validator written in Python, which is slower but gets the job done.

Chapter 6. Plain text Foods files, and choosing which type of Foods file to use

So far, all the files that we have worked with have either been binary files[1] or XML files. Pantry can also store foods in a more traditionally Unix plain text format--that is, in a text file with each row representing a single food and each column representing something about that food. In this chapter you'll learn about the advantages and disadvantages of this file format and how you can use it.

6.1. How Foods Text files work

As you know, a food in Pantry consists of traits, nutrients, and available units. That's a lot of data. The two file formats we have seen so far--the Pantry native file and the Foods XML file--both contain all of this data for each and every food. The Pantry native file is fast, but you can't edit that with your text editor at all. The Foods XML file can be edited with your text editor, but the format is so verbose that you don't ordinarily want to touch it unless you have to--if, for example, you are creating an entirely new food.

The Foods Text file is a little different from the Foods XML and Pantry native files because it does not contain all the data for each and every food. Instead, it relies upon another Foods XML or Pantry native file to contain all the food data. In the Foods Text file, each line corresponds to a particular food. Each line specifies a source file for the food. The source file will be a Foods XML or Pantry native file that will contain the nutrient and unit data for the food.

Pantry must look up the food in the source file. To do this, Pantry uses the food's name. There must be exactly one match for the food name that you specify; therefore, you will find Foods Text files to be easiest to use if you have source files in which every food name is unique. The `master` file meets this criterion; you might also set up a `quick` file that also meets this criterion.

This will all get more clear with examples:

6.2. Foods Text files: an example

Here is an example Foods Text file:

Example 6-1. A sample Foods Text file

```
#source:name:qty:unit:date:meal:group:comment:options
#Options: x=exact match; i=ignore case
master:Milk, reduced fat, fluid, 2% milkfat, with added vitamin A:1.0:cup:2007-06-30:Breakf
master:Papayas:1 1/2:medium
master:blueberries, raw:5:oz:::::i
```

The first thing to notice is that you can begin comments anywhere on a line with a hash mark, `#`. Pantry ignores any text after the comment symbol. Here, the first two lines are handy reminders of the syntax of the Foods Text file.

Colons separate the fields in the Foods Text file. If you need to include a colon or a hash mark in a field, you can precede it with a backslash, `\`. To include a backslash, precede it with a backslash. A single backslash followed by any character other than a backslash, colon, or hash mark generates an error.

Here is what appears in each column of the Foods Text file:

Columns in the Foods Text file

Column: 1
Contents: The source file; that is, the filename of the file holding the food that contains the nutrient and unit data for this food. Pantry searches for this file using the same rules that it uses to search for files specified on the Pantry command line. More specifically, if the file is in a directory in the `path` that you specified in your `.pantryrc.xml`, then you can omit the full path name here.

Column: 2
Contents: The name of the food. Pantry uses this to look up a food in the source file, which then serves as the basis for this food. This is a regular expression, unless `x` is specified in the `options` column. The search is case-sensitive unless `i` is specified in the `options` column. After looking up the food using the contents of this column and the `source` column, Pantry changes the traits of the food according to the following columns.

Column: 3
Contents: The quantity for the food.

Column: 4
Contents: The unit for the new food. The unit is selected from the food's available units. The value in this column is a regular expression, unless `x` is specified in the `options` column, and the search is case sensitive unless `i` is specified in the `options` column.

Column: 5
Contents: The `date` trait for the new food.

Column: 6
Contents: The `meal` trait for the new food.

Column: 7
Contents: The `group` trait for the new food.

Column: 8
Contents: The `comment` trait for the new food.

Column: 9
Contents: These are search options. This column may have zero, one, or two letters. `x` specifies that the search for the food name and for the unit must be an exact match, rather than a regular expression. `i` makes searches case insensitive.

If you only need to use some of the columns, you can leave out trailing columns. The only columns that you must specify are the first two, for the source and the food name.

You may have noticed that there is a column to set most of the food's traits, but there is no column to set the food's `order` trait. This is because Pantry automatically sets the food's `order` trait so that it is equal to the line number of the food in the file, although it is left-padded with zeroes so that it will equal four digits.

6.3. Using Foods Text files

After you have completed your Foods Text file, you can use it just as you would any other Pantry file. Here is an example:

Example 6-2. Using a Foods Text file

```
$ pantry --print traits-units-blank foods.txt
Papayas, raw
Group: Fruits and Fruit Juices
Order: 0004
Refuse: 33 percent Seeds and skin
1 medium (5-1/8" long x 3" dia) (304g)
   small (4-1/2" long x 2-3/4" dia)
   medium (5-1/8" long x 3" dia)
   cup, mashed
   cup, cubes
   large (5-3/4" long x 3-1/4" dia)

Milk, reduced fat, fluid, 2% milkfat, with added vitamin A
Group: Cereal and Milk
Date: 2007-06-30 Meal: Breakfast Order: 0003
1.0 cup (244g)
   quart
   fl oz
   cup

Blueberries, raw
Group: Fruits and Fruit Juices
Order: 0005
Refuse: 5 percent Stems and green or spoiled berries
```

```
5 oz (142g)
   pint as purchased, yields
   cup
```

The nice thing about Foods Text files is that although you can edit them with your text editor, you can also manipulate them using ordinary Pantry commands. For example:

Example 6-3. Changing a Foods Text file

```
$ pantry --name "Apples, raw, with skin" --c-date "April \
> 15" --c-qty 1 --c-unit medium --add foods.txt master
$ cat foods.txt
#source:name:qty:unit:date:meal:group:comment:options
#Options: x=exact match; i=ignore case
master:Milk, reduced fat, fluid, 2% milkfat, with added vitamin A:1.0:cup:2007-06-30:Breakfast:Cereal
master:Papayas:1:medium
master:blueberries, raw:5:oz:::::i
master:Apples, raw, with skin:1:medium (2-3/4" dia) (approx 3 per lb):April 15::Fruits and Fruit Juic
```

As you can see, when you add foods to a Foods Text file using **pantry**, appropriate text is added to the end of the file. However, beware that if you change the name of the food using the `--c-name` option, Pantry will not be able to locate the food in the source file. You can, however, change any of the other traits, although Pantry will override any changes you make to the `order` trait because the `order` trait is automatically set to the line number of the food. Thus the `--auto-order` option has no effect when using Foods Text files.

6.4. Which sort of file should I use?

Thus, there are three different file formats that do similar things: keep Pantry foods. You might be wondering which one you should use.

Each file format has advantages and disadvantages. The Pantry native file is the fastest one for Pantry to work with, but it cannot be edited with a text editor. The Foods XML file is plain-text, but Pantry takes some time to open it and to write changes to it because it is XML. The Foods Text file is also plain-text, and very easy to edit, but it does not hold all the data needed for a food and so it must rely on source files.

I find it best to use a Foods Text file for my diaries. It is remarkably easy to edit, which makes it easy to fix mistakes. It's also easiest to reorder foods in a Foods Text file; then I can easily sort the reports using the `--sort o` option. It's also possible to assign the order trait to foods in other files using the `--c-order` option, but that is more cumbersome.

For files that will contain many foods, such as a `master` file, I use a Pantry native file because it is the speediest. Of course, when adding entirely new foods with new nutrient information, my only choice is to use a Foods XML file.[2] I otherwise avoid Foods XML files because XML is a bit cumbersome to edit.

I also keep a `quick` file for easy access to the foods I use often. I keep this in a Pantry native file, but if your `quick` file is relatively short (under a couple of hundred foods, say) there's no reason not to use a Foods Text file for this (except for the fact that you'll have to make sure it stays harmonized with the source file--that is, you have to ensure that the foods that the Foods Text file looks up in the source file remain in the source file.)[3]

Notes

1. The distinction between binary files and plain-text files is quite arbitrary. It is fair to say that there is no such thing as plain text (http://www.joelonsoftware.com/articles/Unicode.html). But for our purposes we will call anything that you usually edit in a text editor "plain text".
2. I have considered making a way to add entirely new foods using the Pantry command-line interface, but I don't see how much of a benefit this would be.
3. Here is as good a place as any to point out that there is yet another Pantry file format: the Foods Zip file. This is what I use to import the SR data from the Zip file that USDA provides. If you are curious, you can download (http://www.ars.usda.gov/Services/docs.htm?docid=13746) the full ASCII version and run something like **`pantry --add master sr19.zip`** to make your very own `master` file. I can't guarantee that this will work with any releases subsequent to SR 19, and it takes lots of RAM too (about 200 MB, maybe.)

Chapter 7. Adding recipes

So far, you know how to add your own foods to Pantry. However, to add a food, you have to know what its nutrient content is. If you have a food that is a mixture of several different foods, then making a recipe can help. You'll learn how to do that in this chapter.

7.1. When to use recipes

The basic building block of Pantry is the food. Each food has traits, nutrient content, and available units. In Pantry, you can combine several foods into one single food using recipes. Thus, for example, you can combine corn meal, flour, water, baking powder, sugar, salt, and eggs using a recipe. The result will be a single food that you can call corn bread. This new food will have traits, available units, and nutrients, just like any other food in Pantry.

After you create a recipe in Pantry, Pantry knows only the traits, available units, and nutrients of your new food. For example, after you create your corn bread recipe, Pantry does not know what the ingredients of corn bread are. It knows only the traits and the nutrient breakdown of corn bread. You can then use corn bread just as you would use any other food. Because a recipe is a combination of other foods, each ingredient in the recipe must come from an existing Pantry file.

Sometimes using a recipe is overkill. For example, say you have a few foods that you always eat together in a certain quantity. It can be much quicker and easier to simply give these foods a common group trait. For example, I often eat two ounces of cold cereal with one cup of milk. I could create a recipe for these foods. Instead, I have simply added both of these foods to my `quick` file, and I have changed the `group` trait of both foods to *`Cereal and Milk`*. Then, when I want to record in my diary that I have eaten these two foods, I simply issue **`ab -g 'cereal and milk'`**.

Thus you will find recipes most useful in two instances. First, recipes are handy if you have several foods that you wish to combine into one food. Then the nutrient breakdown of all the foods will appear together in your reports. Second, you have to use recipes if you wish take several foods, combine them, and subdivide the results. For example, if I wanted to eat an entire 9x9-inch dish of corn bread, perhaps I could treat corn bread in Pantry the same way I record my cereal and milk. However, because I want to divide the corn bread into multiple portions and eat them separately, I must use a recipe.

7.2. Creating a recipe

You will recall that there are several types of Pantry files. You have already learned about two: the Pantry native file, and a Foods XML file. You create recipes using a third type of Pantry file: a Recipe XML file. Unlike Pantry native files and Foods XML files, Recipe XML files are read only. That is, though Pantry can read foods created in Recipe XML files, Pantry cannot add foods to Recipe XML files.

7.2.1. Creating a recipes file and its root element

To create a Recipe XML file, start your favorite text editor and open a text file. For the purposes of our example we'll use `recipes.xml`, though you can create any filename that ends with `.xml`.

The root element of every Recipe XML file is simply `recipes`. The `recipes` element has a single attribute, `foodSource`. Every recipe in Pantry is a new food that is comprised of several different foods. The `foodSource` attribute specifies which file contains the foods that will be used to make the recipes in this Recipe XML file. This file may be a Pantry native file or a Foods XML file or even a Foods Text file.[1] When specifying `foodSource`, it is easiest to use an absolute path--that is, one that begins with a slash.[2] Most likely you will want to use the `master` file, as we will do in this example.

Example 7-1. A `recipes.xml` file with an empty `recipes` element

```
<recipes foodSource='/home/massysett/pantry/master'>
</recipes>
```

7.2.2. Creating the `recipe` element

Next you will need to create a `recipe` element. This element will contain all the information needed for a single recipe. Remember that each recipe will ultimately create a new food. You use the attributes of the `recipe` element to specify the traits of the new food you are creating. Thus, the `recipe` element has several attributes: name, group, refuse, refDesc, date, meal, qty, comment, unit. Of all these attributes, you are required to set only the `unit` and `qty` attributes. You can leave as many of the other ones out as you wish, though ordinarily you will set the `name` and `group` attributes while leaving many of the others blank.

As with Foods XML files, you must set the `unit` attribute to one of the available units for the recipe. Pantry will automatically create the *`g`*, *`oz`*, and *`lb`* units for you. You can also have Pantry create a *`serving`* unit, as we will discuss shortly.

For our example, we will create a recipe with the name *`Easy Corn Bread`*. As you will see, the new recipe has its unit set to *`serving`*; shortly we will see exactly how this works. You will also notice that many of the attributes for the various traits, such as `date` and `meal`, are not set at all; Pantry will set such traits to the empty string.

Example 7-2. Creating the `recipe` element

```
<recipes foodSource='/home/massysett/pantry/master'>
<recipe name="Easy Corn Bread"
    group="Baked Products" pctRefuse=" refDesc="
    qty='1' unit='serving'>
</recipe>
</recipes>
```

7.2.3. Setting the yield of a recipe

Next you will need to set the yield of your recipe. The `yield` element has two attributes: `grams` and `servings` The `servings` attribute is easier to understand, so we will discuss it first.

7.2.3.1. The `servings` attribute

Often when you make a recipe, you split it into an even number of servings. If that is the case for your recipe, you may enter an appropriate value for the `servings` attribute. Pantry will automatically create a *`serving`* available unit for the food in this case.

For our corn bread example, we will say that the recipe makes nine servings.

7.2.3.2. The `grams` attribute

In order for Pantry to accurately calculate the nutrient content of a particular amount of food, it needs to know what the total mass of your completed recipe is, after it has been fully prepared (that is, after cooking, baking, mixing, etc., as the case may be.)

Pantry can "guess" what the total mass of your completed recipe is. Pantry guesses by adding up the mass of all the ingredients you entered for the recipe. To have Pantry guess for you, enter *`guess`* for the `grams` attribute. However, you will often find that the completed recipe weighs significantly less than the mass of all the ingredients. This is because a lot of water often evaporates as you cook or bake foods. Thus, for the most accurate results, you may enter the mass of your completed recipe so that Pantry may take it into account. To do this, enter the appropriate value (in grams) for the `grams` attribute.

However, if you have entered a `servings` attribute, and you will use *only* the *`serving`* available unit when you use the recipe, then entering a `grams` attribute will do you no good. This is because Pantry will calculate each serving so that it is the appropriate fraction of the total, but the actual total mass will not matter.

I always cut my cornbread into nine neat squares, and I always eat only one square at a time. Thus, since I will use only the *`servings`* available unit for the corn bread, I will simply enter *`guess`* for the `grams` attribute.

7.2.3.3. Contents of the `yield` element

Finally, you may enter any text you wish between the opening and closing tags of the `yield` element. This text can be used to describe the yield of the recipe. Pantry ignores this text, but it is useful because you might find yourself using the XML for other purposes (perhaps, for instance, to print your recipe out for use in the kitchen.)

I make my corn bread in a 8x8-inch dish, so I will make an appropriate entry for the `yield` element.

7.2.3.4. A complete `yield` element

Here is an example `yield` element:

Example 7-3. Setting the recipe yield

```
<recipes foodSource='/home/massysett/pantry/master'>
<recipe name="Easy Corn Bread"
    group="Baked Products" pctRefuse=" refDesc="
    qty='1' unit='serving'>

    <yield grams='guess' servings='9'>
        One square pan
    </yield>
</recipe>
</recipes>
```

7.2.4. Setting the ingredients of a recipe

No recipe is complete without ingredients. Each single ingredient is represented by a `ingredient` element. The `ingredient` element has four attributes: `name`, `qty`, `unit`, and `comment`.

First, as you will recall, each ingredient comes from the Pantry file given in the `foodSource` attribute of the `recipes` element. The `name` attribute of the `ingredient` is an exact match of the food you wish to use from the `foodSource` for this ingredient. You must exactly match every space, and you must exactly match the mixture of upper- and lower-case letters, as well.

The second attribute, `qty`, is simply the quantity of the ingredient. This must be an integer or floating-point number.

The third attribute, `unit`, is an exact match of the available unit you wish to use for this ingredient. As with the `name` attribute, this is case-sensitive.

Finally, the last attribute, `comment`, is optional. Pantry ignores this attribute, but you might find it handy for you own purposes.

Here is our corn bread example, complete with all its ingredients:

Example 7-4. A recipe with its `ingredients`

```
<recipes foodSource='/home/massysett/pantry/master'>
<recipe name="Easy Corn Bread"
    group="Baked Products" pctRefuse=" refDesc="
    qty='1' unit='serving'>

    <yield grams='guess' servings='9'>
        One square pan
    </yield>

    <ingredient name='Wheat flour, white, all-purpose, enriched, unbleached'
        qty='5 5/8' unit='oz' comment=" />
    <ingredient name='Cornmeal, whole-grain, yellow'
        qty='4.125' unit='oz' comment=" />
    <ingredient name='Sugars, granulated'
        qty='1 3/4' unit='oz' comment=" />
    <ingredient name='Leavening agents, baking powder, double-acting, straight phosphate'
        qty='2' unit='tsp' comment=" />
    <ingredient name='Salt, table'
        qty='1/2' unit='tsp' comment=" />
    <ingredient name='Milk, reduced fat, fluid, 2% milkfat, with added vitamin A'
        qty='1' unit='cup' comment=" />
    <ingredient name='Oil, canola and soybean'
        qty='1/4' unit='cup' comment=" />
    <ingredient name='Egg, whole, raw, fresh'
        qty='1' unit='large' comment=" />
</recipe>
</recipes>
```

7.2.5. Setting the available units for a recipe

Finally, you may set the available units for the food. Pantry automatically creates the *`g`*, *`oz`*, and *`lb`* units for you; in addition, if you specified a *`servings`* attribute to the `yield` element, as we discussed above, then Pantry will also automatically create a *`serving`* available unit. If you wish to create any additional available units, you will need to know the weight (in grams) of that unit. If for instance you are entering a recipe for cookies and you want to create an avaiable unit for one cookie, you'll need to know the weight in grams of one cookie.

To create available units, create one or more `unit` elements. Each `unit` element will have both `name` and `grams` attributes. As I stated above, I always eat one serving of corn bread, so I do not have any use for the `units` element here. However, for the sake of illustration, we'll say that a *`large piece`* of corn bread weighs 70 grams.

Example 7-5. A complete new recipe

```
<recipes foodSource='/home/massysett/pantry/master'>
<recipe name="Easy Corn Bread"
    group="Baked Products" pctRefuse=" refDesc="
    qty='1' unit='serving'>

    <yield grams='guess' servings='9'>
        One square pan
    </yield>

    <ingredient name='Wheat flour, white, all-purpose, enriched, unbleached'
        qty='5.625' unit='oz' comment=" />
    <ingredient name='Cornmeal, whole-grain, yellow'
        qty='4.125' unit='oz' comment=" />
    <ingredient name='Sugars, granulated'
        qty='1.75' unit='oz' comment=" />
    <ingredient name='Leavening agents, baking powder, double-acting, straight phosphate'
        qty='2' unit='tsp' comment=" />
    <ingredient name='Salt, table'
        qty='.5' unit='tsp' comment=" />
    <ingredient name='Milk, reduced fat, fluid, 2% milkfat, with added vitamin A'
        qty='1' unit='cup' comment=" />
    <ingredient name='Oil, canola and soybean'
        qty='.25' unit='cup' comment=" />
    <ingredient name='Egg, whole, raw, fresh'
        qty='1' unit='large' comment=" />
    <unit name='large piece' grams='70' />
</recipe>
</recipes>
```

7.3. Using recipes

Once created, you may use a Recipe XML file just as you would use any other Pantry file. There is one exception to this though: Recipe XML files are read-only. Thus, although you may search Recipe XML files just as you would any other Pantry file, you may not add foods to them using Pantry. (You may of course add foods using your text editor, which is what we have been discussing in this section.)

When Pantry reads in recipes from a Recipe XML file, it first finds all the ingredients in the specified `foodSource` file. It then calculates the total nutrient breakdown of all the ingredients, before dividing the recipe into portions using any information given in the `yield` element. Pantry then creates a food in the buffer that has the appropriate traits, available units, and nutrients. After the food is in the buffer, it is indistinguishable from other Pantry foods. This means that Pantry no longer knows what ingredients the food contains, though this information remains in the Recipe XML file.

Doing all this searching and calculating takes a little while. The time is not noticeaeble for a few recipes, but it could be noticeable for several recipes. In addition, you'll probably find it more convenient if your recipes are in a file with all the other foods you use. Thus, you will probably find it most convenient to add your new foods to a Pantry native file, such as the `master` file or a `quick` file.

Example 7-6. Using a Recipe XML file

```
$ pantry --name "Easy Corn Bread" --print traits-units-nuts \
> recipes.xml
Easy Corn Bread
Group: Baked Products
1 serving (76g)
   serving
Nutrient                  Amount          %G     %TOT
-------------------------------------------------------
Calories                  208  kcal        10     100
Total Fat                 8    g           12     100
Saturated Fat             1    g           5      100
Cholesterol               26   mg          9      100
Sodium                    234  mg          10     100
Total Carbohydrate        31   g           10     100
Dietary Fiber             1    g           6      100
Sugars                    7    g           NG     100
Protein                   4    g           9      100
Vitamin A                 106  IU          2      100
Vitamin C                 0    mg          0      100
Calcium                   114  mg          11     100
Iron                      1    mg          8      100
```

Example 7-7. Adding a recipe to a Pantry native file

```
$ pantry --add master recipes.xml
$ pantry --name "Easy Corn Bread" --print names master
Easy Corn Bread
```

7.4. Validation of Recipes XML files

As with Foods XML files, Pantry automatically validates Recipes XML files and will show you an error message if the XML is invalid. To see the DTD that Pantry uses for this purpose, run **`pantry --dump recipesDTD`**. For more details on validation, see Section 5.4.

Notes

1. Strictly speaking, it can be a Recipe XML file as well, but it is unlikely you would want to do such a thing.

2. You can use a relative path if you want. Such a path will be resolved relative to your current directory when you execute the Pantry command; the directory of the Recipe XML file is not relevant. This is confusing, which is why I recommend specifying an absolute path.

Appendix A. Reference pages

This appendix contains reference pages for the commands that Pantry uses. They are identical to the man pages.

pantry

Name

`pantry` — nutrient analyzer

Synopsis

pantry [options...] [*`file`*...]

pantry [--dump *`dumpable`*]

Description

pantry copies foods from *`FILE`*s into a buffer. All foods are copied, unless `SEARCH OPTIONS` are specified, in which case only matching foods are copied. `SEARCH OPTIONS` are cumulative. **pantry** then changes every food in the buffer using any `CHANGE OPTIONS` specified.

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

If `--print` *`REPORT`* is specified, buffer is printed using *`REPORT`* If `--nutrient-list` *`NUTRIENT-LIST`* is specified, buffer is printed using *`NUTRIENT-LIST`*; otherwise, the default nutrient list is used. Buffer is unsorted unless `--sort TRAITS` is specified.

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

Options

Pantry uses Perl compatible regular expressions.

Search options

`-n` *regexp*
`--name` *regexp*

Include foods whose `name` trait matches *regexp*.

`-g` *regexp*
`--group` *regexp*

Include foods whose `group` trait matches *regexp*.

`-d` *regexp*
`--date` *regexp*

Include foods whose `date` trait matches *regexp*.

`-m` *regexp*
`--meal` *regexp*

Include foods whose `meal` trait matches *regexp*.

`-u` *regexp*
`--unit` *regexp*

Include foods whose `unit` trait matches *regexp*.

`-c` *regexp*
`--comment` *regexp*

Include foods whose `comment` trait matches *regexp*.

`-q` *number*
`--qty` *number*

Include foods whose `qty` trait matches *number*.

`-o` *regexp*
`--order` *regexp*

Include foods whose `order` trait matches *regexp*.

Change options

`-N` *string*
`--c-name` *string*

Change `name` trait to *string*.

`-G` *string*
`--c-group` *string*

Change `group` trait to *string*.

`-D` *`string`*
`--c-date` *`string`*

Change `date` trait to *`string`*.

`-M` *`string`*
`--c-meal` *`string`*

Change `meal` trait to *`string`*.

`-U` *`regexp`*
`--c-unit` *`regexp`*

Search each food's available units using *`regexp`*. If there is exactly one match, set `unit` trait to that match; otherwise, print warning, remove food from buffer, and do not process the `--delete` or `--edit` options for this food.

`-C` *`string`*
`--c-comment` *`string`*

Change `comment` trait to *`string`*.

`-Q` *`number`*
`--c-qty` *`number`*

Change `qty` trait to *`number`*.

`-O` *`string`*
`--c-order` *`string`*

Change `order` trait to *`string`*.

`--by-nut` *`regexp amount`*

Change quantity of food so that amount of nutrient matched by *`regexp`* equals *`amount`*

`-K` *`amount`*

Same as `--by-nut` *`Calories amount`*

`--refuse` `-R`

Reduce qty of every food in buffer by its corresponding refuse trait

`--auto-order`

When adding each food to files specified with `--add`, **pantry** will search the file 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, removes the last digit, and increment the result by one. The result will be left-padded with zeroes so that it is four characters long. **pantry** will then change the trait of the food to the result before adding it to the file.

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

`--auto-order` has no effect when adding foods to Foods Text files.

Print options

`-p` *report*
`--print` *report*

Print buffer using *report*.

`-l` *nutrient-list*
`--nutrient-list` *nutrient-list*

use *nutrient-list* when printing report

`-s` *traits*
`--sort` *traits*

Sorts foods by trait when printing a report. Specify traits by their first letter; for example, to sort by name, date, and meal, specify `--sort ndm`. Use lower-case letters to sort in ascending order; use upper-case letters to sort in descending order.

Other general options

`-i`
`--ignore-case`

Make all `Search options`, and the `--c-unit` option, case insensitive

`-x`
`--exact-match`

Arguments to `Search options`, to `--c-unit` option, and to `--by-nut` option must exactly match food traits or nutrient names, rather than using regular expressions

`-a` *file*
`--add` *file*,

Add buffer to *file*. To specify multiple files, use multiple `--add` options, e.g. **`--add file1 --add file2`**.

`--edit`

Delete original foods from corresponding source files, and add changed foods to corresponding source files

`--delete`

Delete original foods from corresponding source files

Other options

`--force-valid`

Ordinarily **pantry** automatically validates all XML files unless they are over 300 kilobytes in size; use this option to validate all XML files regardless of their size.

`--skip-valid`

Do not validate any XML files

`-h`
`--help`

Show brief help message and exit

`--version`

Print version information and exit

`--dump`

Display an internal Pantry variable and exit (see DUMP OPTIONS below)

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, specify each report name, separated by dashes; for example, `--print names-nuts`. Reports may also be specified with an unambiguous specification of the first letters of the report, such as `--print na-nu`.

The following reports are available:

Food reports

`names`

Food names

`traits`

Food traits

`units`

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

`measures`

Like *`units`*, but also shows the gram weight of every unit.

nuts

Nutrients. For details of how **pantry** decides which nutrients to print, and what all the columns in this report mean, see .

blank

A blank line

paste

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

txt

Each food in a format that may be pasted into a Foods Text file.

Summary reports

sum

Nutrient total of all nutrients in the buffer

groups

Group names, the number of foods in each group, and the total number of foods in the buffer

list

The nutrient list, along with its goals

Nutrient lists

A nutrient report has four columns. The first shows the name of the nutrient. The second column shows the amount (both the numeric amount and the units.) The third column shows this nutrient's percent of a nutrient goal, or blank if that nutrient has no goal. The fourth column shows this nutrient's percentage of the total nutrients in this buffer. The fourth column is not present in *sum* reports.

The `nutrient-list` parameter determines which nutrients are shown and in what order, as well as determining what goals are used to calculate the third column. The user may configure her own `nutrient-lists` in pantryrc.xml(5). The following default `nutrient lists` are available:

facts

Mimics the USA "Nutrition Facts" panel.

dv

Nutrients for which there is a USA FDA Daily Value.

all

All nutrients.

short

Only *Calories*, *Total Fat*, *Total Carbohydrate*, and *Protein*.

Dump options

`--dump` takes a single argument, which will print one of the following **pantry** internal variables. After printing the variable, **pantry** will exit.

version

Version and copyright information; same as `--version`.

foodsDTD

DTD used to validate Foods XML files

recipesDTD

DTD used to validate Recipes XML files

rcDTD

DTD used to validate `pantryrc` files

config

All configuration variables after the `pantryrc` is processed

Bugs

Documentation is still incomplete.

Under certain circumstances, consumes about 100MB of RAM, though briefly, when buffer totals about 7,000 foods (the number in the USDA database.)

Files must be disk files; more specifically, they must supportlseek(2). Using other files (such as stdin or stdout) will probably crash Pantry.

Report additional bugs to <`pantry-users@lists.sourceforge.net`>.

Version

This manual page written for Pantry version 11.

See also

pantry-addTo(1), pantryrc(5)

Appendix B. Pantry's birth

I have used computers to analyze recipes and track food intake for years now. Years ago I still used Microsoft Windows. There are some good proprietary programs in Windows that are useful for food analysis, but even the best aren't extremely well-documented. So once I discovered that the USDA food database is freely available, Microsoft Access seemed a natural way for me to develop my own application.

My Access application went through many iterations. Then I switched to Linux, which taught me one big lesson: never rely on expensive proprietary software. I poked around to see if there were any good Linux food analysis programs. The only well-developed one I could find, NUT, was a console program. At the time this scared me, so I just did not have a food analysis program for quite some time.

As I gained more experience with Linux, I discovered that there were Access-like applications for it, such as OpenOffice Base (which is still fairly new) and Kexi. But I did not trust these applications much: I generally found that huge office-suite programs for Linux are not very well documented and tend to be bloated. True, Access and Microsoft Office are bloated too, but at least there are tons of tutorials available on the Net. So I never touched OpenOffice Base or Kexi.

Though the Linux console scared me at first, as I used Linux more and more I realized how powerful and simple the console is. I came to appreciate its flexibility and even its beauty. So I took another look at NUT. Though I realized it is an excellent program, I couldn't quite get along with it. It ran in an console, but it was menu-driven and seemed to have arbitrary limits that I didn't like. There are Web-based programs for this purpose too, but I always find them cumbersome to use and poorly designed.

Then I slowly learned of command-line tools such as **cut** and **join**. I was surprised to learn that one can build a relational database management system (http://www.rdb.com/) using shell tools. I also was inspired by other command-line oriented tools that did things that I previously thought would be possible to do well only in a GUI. An excellent example of this is Ledger (http://www.newartisans.com/ledger.html), a program that I now find indispensable for tracking my finances. So I decided to try writing a shell-based food analysis program; I called the program bashfood and, later, Shellfood.

Shellfood worked okay when it was small, but as it grew it became unmanageable. Shells are great for certain things, but building large programs is not one of those things. Eventually it became apparent to me that I would need to switch to a different language. Python seemed a natural choice, as it is often recommended for beginners. It is also a full-fledged language used by professionals--Linux always impresses me because tools that cost several hundreds of dollars for Windows are available free on Linux--better tools, too.

So, the result of several months of work is Pantry. It has evolved constantly as I have used and refined it. It works even better than I had envisioned, though others might find it hopelessly complicated. We'll see. Let me know what you think! Report bugs, too--I know they must be in here. Send emails to

pantry-users@lists.sourceforge.net.

If you like Pantry, be sure to check out other command-line programs like Ledger (http://www.newartisans.com/ledger.html) and Todo.txt (http://todotxt.com/) because they can make your life a lot easier. Good text editors like Vim and Emacs are indispensable, as is a nice shell like Zsh.