# GNU units program

someonesdad1@gmail.com 6 Jan 2011, updated 5 May 2012

Unit errors have cost lots of money and lives over the centuries. One well-known example is a Mars spacecraft worth hundreds of millions of dollars that was lost because of an embarrassing mistake with units.

The GNU units program can help with unit conversion problems; it's a text-based program that runs in a console. I'll demonstrate a few of its features here. It's one of the most-used programs on my computer.

I'll assume you can figure out how to get the program. You may have to do the usual configure/make dance to build it from scratch. If you're on a Windows computer, you can build it with the MinGW/MSYS stuff or use a cygwin environment. There's a project on SourceForge that claims to offer a Windows build, but I haven't tried it. There is a Java GUI version of the units program here, but I haven't used it.

The bread-and-butter functionality of the units program is to convert between conformable units of measure (the output you see here uses the `--verbose` option):

```
You have: 25.4 cm
You want: in
        25.4 cm = 10 in
        25.4 cm = (1 / 0.1) in
```

Here, we entered `25.4 cm` and asked that it be converted to inches.

The second printed number is the inverse of the conversion. This is useful in some situations. For example,

```
You have: grains
You want: pounds
        grains = 0.00014285714 pounds
        grains = (1 / 7000) pounds
```

(Note we entered just the unit, which implies a leading coefficient of unity.) This says that 0.14 millipounds is about one grain. The inverse relationship tells us that there are exactly 7000 grains in one pound. From here on, I'll leave out the inverse conversion. I read somewhere that someone once mistook a prescription written in grains (abbreviated gr) to mean a dose in grams. The medicine was a phenobarbital and the effects could be deadly because the patient could get 15 times the desired dose:

```
You have: g
You want: grain
        g = 15.432358 grain
```

You can see the base SI units that make up a composite unit:

```
You have: T
You want:
        Definition: Tesla = Wb/m^2 = 1 kg / A s^2
```

This can help you with dimensional analysis tasks -- you can use it to see the base SI units making up any expression of composite units.

The following units are also defined in the `units.dat` file:

```
LENGTH                  MONEY                   INDUCTANCE
AREA                    FORCE                   FREQUENCY
VOLUME                  PRESSURE                VELOCITY
MASS                    STRESS                  ACCELERATION
CURRENT                 CHARGE                  DENSITY
AMOUNT                  CAPACITANCE             LINEAR_DENSITY
ANGLE                   RESISTANCE              VISCOSITY
SOLID_ANGLE             CONDUCTANCE             KINEMATIC_VISCOSITY
```

These are intended to help with dimensional analysis. For example:

```
You have: VELOCITY
You want:
        Definition: LENGTH / TIME = 1 m / s
```

Another technique is to answer the `You want:` prompt with a question mark. The units program will then show you a list of all the units it knows about that you can convert the given unit to. Thus, if you enter the unit `T` for `Tesla`, you'll see the equivalent units of `gauss`, `abvolt*sec/cm^2`, and `Wb/m^2`. You can also type `help T` and your pager will be indexed at the definition line in the `units.dat` file; this is useful because this file typically has numerous comments about the units, especially the obscure ones.

The rules to enter expressions with units are straightforward. Multiplication is indicated by the operator `*` or simply one or more blank spaces. Exponentiation is indicated by the `^` operator, although an integer exponent for a unit can just have the integer cuddled after the unit (fractional exponents are not allowed). Thus, `mm^2` or `mm2` indicate square `mm`. The SI prefix has a higher precedence than the exponent; thus, `mm^2` means `(mm)^2`, not `m(m^2)`. Division is indicated by `/`.

Multiplication has higher precedence than division. This is contrary to most programming languages which make them equal in precedence and then resolve things using associativity. However, it helps deal with sloppy expressions like `g/kA*day`. Usually, when a person uses such an ambiguous expression, he means `g/(kA*day)`. The GNU units program handles this expression as you'd expect. However, you'll also see people use expressions like `m/s/s`, which is ambiguous: it can mean either $m/s^2$ or `m`. Interpreting it as `m/(s/s)` is a bit pointless, so the GNU units program interprets it as the prior expression.

You can also write unit expressions where the units are all on one line (no division symbol) and utilize positive or negative exponents. Thus

```
You have: m s^-1 jansky^-2 T3
You want:
        Definition: 1e+52 kg m / A^3 s^3
```

Note the unary minus operator can be used here without parentheses. However, you must be careful to use the exponentiation operator `^`, as omitting it means something entirely different:

```
You have: s-1
You want:
        Definition: 1 s
You have: s(-1)
You want:
        Definition: -1 s
You have: s^-1
You want:
        Definition: 1 / s
```

You can add conformable units:

```
You have: 8 weeks + 14 days + 17 hours + 22 minutes
You want: seconds
        8 weeks + 14 days + 17 hours + 22 minutes = 6110520 seconds
```

Arc measure is slightly different:

```
You have: 8 degrees + 14 arcminutes + 22 arcseconds
You want: microradians
        8 degrees + 14 arcminutes + 22 arcseconds = 143805.43 microradians
```

This shows that you can use SI prefixes as needed. You can also use sexagesimal measure:

```
You have: cos(44 deg + 59' + 60")
You want:
        Definition: 0.707107
```

which shows `'` is defined as `arcminute` and `"` is defined as `arcsecond`.

Fractions can be denoted with the `|` symbol. As mentioned, division is indicated by the `/` symbol; however, you sometimes must be careful. For example `1/2 m` doesn't mean half a meter as most of us would expect -- it means `0.5` $m^{-1}$ (because multiplication is higher precedence than division, the

expression is parsed as `1/(2*m)`). Because of this, the `|` symbol can be used for division, as it has higher precedence than the multiplication operator. Then `1|2 m` gives you what you expect. `|` is also higher precedence than the exponentiation operator `^` so that `1|2^1|2` gives you what you expect.

The units program is handy for back-of-the-envelope calculations. For example: suppose a room measuring $3\frac{1}{2}$ m by $88\frac{3}{4}$ ft by 1260 inches was filled with water and drained out in 30 days, 13 hours, and 24 minutes. What was the average flow rate in cubic furlongs per fortnight?

```
You have: (3+1|2) m (88+3|4) ft 1260 in / (30 days + 13 hours + 24 minutes)
You want: furlongs3/fortnight
```

The answer is 0.00017051976. In case you don't read Jane Austen novels, a fortnight is two weeks. A furlong is 40 rods; a rod is 5.5 yards; thus, a furlong is 220 yards or an eighth of a statute mile. In case you're wondering, `0.00017051976 furlongs³/fortnight` is a little over a liter per second.

You can do basic arithmetic calculations:

```
You have: (87.3*1.9 + 48.5)/69.1
You want:
        Definition: 3.1023155
```

The output is limited by default to 8 significant figures. Use the `-o` option to change the output formatting (you can get the full floating point resolution this way if you want it). I use a default of 6 significant figures.

The program also has some built-in elementary functions:

```
You have: 336 inches tan(17 degrees + 22 arcminutes)
You want: m
        336 inches tan(17 degrees + 22 arcminutes) = 2.6690671 m
```

The other functions are `acos`, `asin`, `atan`, `cos`, `cuberoot`, `exp`, `ln`, `log`, `log2`, `sin`, and `sqrt`. This lets the program be a basic command-line calculator -- just remember all trig arguments need to be in radians or you need to write expressions like `cos(30 deg)`.

The program supports a number of nonlinear units such as temperature and American Wire Gauge:

```
You have: wiregauge(24)
You want: mm
        wiregauge(24) = 0.51055923 mm
You have: 1 mm
You want: wiregauge
        1 mm = wiregauge(18.201919)
You have: tempC(40)
You want: tempF
        tempC(40) = tempF(104)
```

These conversions use a functional notation. Don't confuse the above temperature conversion (which uses the expression `(9/5)*40 + 32`) with the following conversion of a temperature difference:

```
You have: 40 degC
You want: degF
        40 degC = 72 degF
```

which is gotten simply by multiplying 40 by 9/5:

```
You have: 40 9|5
You want:
        Definition: 72
```

You can define your own functions in the `units.dat` file. For examples, look at the definitions of the following functions:

```
pH(x)
tempC(x) and tempF(x)
wiregauge(x)
circlearea(r)
spherevolume(r)
```

```
spherevol(r)
square(x)
```

You can define piecewise linear functions that then subsequently use linear interpolation to find the values. See the definition of the British Standard Wire Gauge function `brwiregauge` for an example.

You can extract roots as in the following example of a fifth root (the fraction operator `|` has the highest precedence to make such expressions possible):

```
You have: 2|3^1|5
You want:
        Definition: 0.92210791
```

There are a number of constants defined in the program that can aid calculations (see the man page for more details). For example, you can calculate the height of a column of water that would have a weight that would yield a pressure of 1 atmosphere at the bottom of the column:

```
You have: atm
You want: m water
        atm = 10.33227453 m water
You have: foot water
You want: psi
        foot water = 0.4335275 psi
```

The built-in constants are:

| | |
|---|---|
| `pi` | Ratio of a circle's circumference to diameter |
| `c` | Speed of light |
| `e` | Charge on an electron |
| `force` | Acceleration of gravity |
| `mole` | Avogadro's number |
| `water` | Pressure per unit height of water |
| `Hg` | Pressure per unit height of mercury |
| `au` | Astronomical unit |
| `k` | Boltzmann's constant |
| `mu0` | Permeability of vacuum |
| `epsilon0` | Permittivity of vacuum |
| `G` | Gravitational constant |
| `mach` | Speed of sound |

Thus, you can verify a liter of water has a mass of 1 kg:

```
You have: 1 l water/force
You want: kg
        1 l water/force = 1 kg
```

The units program uses a data file `units.dat` that contains all the unit definitions. You can edit this file to add units of your own choice and remove units you don't wish to support. The default file supports many archaic units (much of the work of the units program's author and other contributors was tracking down credible definitions of these archaic units).

You can also create other `units.dat` files to do special things. For example, if you were doing some engineering work and wanted to check some US engineering calculations in the foot-pound-second system, you could define a `units.dat` file that would cause all the calculations to output in the requisite units. Or, you might want to define things like the speed of light and Planck's constant to be unity and do various calculations with those units (so-called "natural units").

If you edit the `units.dat` file, it's important to run the program with the `-c` option. This checks that 1) all the definitions reduce to primitive units (those that aren't defined in terms of any other units) and 2) there are no circular definitions; it prints out any problems found.

As an example of why you might want to edit the `units.dat` file, I use the `micro` SI prefix a lot, but prefer to use `u` for it. The default `units.dat` file defines `u` to be the atomic mass unit. It was an easy task to

change all the existing occurrences of `u` to `amu`, comment out the definition of `u`, then add the line

```
u-          micro
```

to let me use `u` for `micro`. Another example: US-sized number screws are often labeled as e.g. "No. 4". I wanted to be able to use no(4) to denote the associated linear dimension; it was easy to copy and modify the line that defined the function `screwgauge`.

Another feature is that you can use the SI prefixes independently of the units. Thus, for example, you could regress to 50+ years ago in electronics and use μμF for capacitance as follows

```
You have: u uF
You want:
        Definition: 1e-12 A^2 s^4 / kg m^2
```

whereas the program will refuse to recognize a unit expression like `micromicroF`. However, be careful, as there are quirks:

```
You have: c m
You want:
        Definition: 2.99792e+08 m^2 / s
```

In case you thought you would get cm units (which you will if the `c` and `m` are cuddled), `c` by itself means the speed of light.

The atomic masses of the elements are included in the program, although they're dimensionless numbers instead of e.g. `g/mol` like you'd expect. To get them, just use the name of the element. Example: we can figure out how many moles of sulfur are in a tank containing 8378 kg of sulfur:

```
You have: 8378 kg / sulfur (g/mol)
You want: mol
        8378 kg / sulfur (g/mol) = 261274 mol
```

One nice feature of open source software is that you can change its behavior should you wish. For the units program, I prefer to be able to type a `q` in when being prompted for a number with a unit and have the program exit. It was easy to add two lines to the program's `unit.c` file to add this behavior.

The GNU units program respects localization issues. An example is the gallon -- if the locale is `en_GB` (Great Britain), the definition of the gallon is changed to the British gallon, `brgallon`.

I use the following shell function to invoke the units program:

```
u()
{
    typeset digits=${1-6};
    typeset data="$xx/units.dat";
    typeset fmt="%.${digits}g";
    $xx/units_.exe -o "$fmt" --verbose -f $data
}
```

(the `xx` variable holds a directory name where I keep such things). This lets me change the numerical resolution by including the new resolution on the command line, although 6 figures is adequate for virtually all the things I do.

Note: I'm using version 1.80 of the units program. This version is roughly a decade old, so if you see different results, you may first want to suspect that they're due to differences in version. (I continue to use this old version because the 1.88 version doesn't operate correctly on my older computer.)