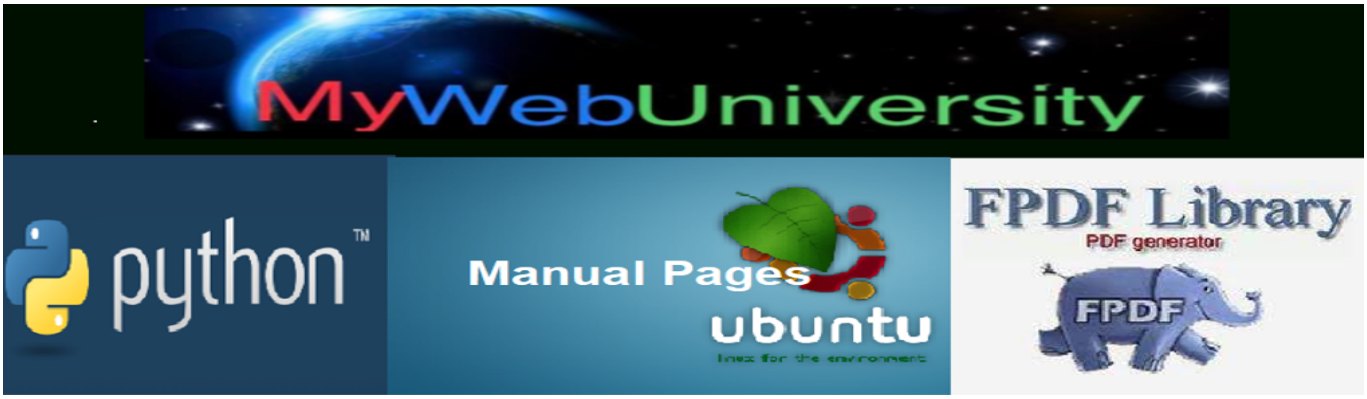

Rocky Enterprise Linux 9.2 Manual Pages on command 'Sort::Naturally.3pm'

$ man Sort::Naturally.3pm

NAME

Sort::Naturally -- sort lexically, but sort numeral parts numerically

SYNOPSIS

```
@them = nsort(qw(
 foo12a foo12z foo13a foo 14 9x foo12 fooa foolio Foolio Foo12a
));
print join(' ', @them), "\n";
```

Prints:

```
9x 14 foo fooa foolio Foolio foo12 foo12a Foo12a foo12z foo13a
```

(Or "foo12a" + "Foo12a" and "foolio" + "Foolio" and might be switched, depending on your locale.)

DESCRIPTION

This module exports two functions, "nsort" and "ncmp"; they are used in implementing my idea of a "natural sorting" algorithm. Under natural sorting, numeric substrings are compared numerically, and other word-characters are compared lexically.

This is the way I define natural sorting:

? Non-numeric word-character substrings are sorted lexically, case-insensitively: "Foo" comes between "fish" and "fowl".

? Numeric substrings are sorted numerically: "100" comes after "20", not before.

? \W substrings (neither words-characters nor digits) are ignored.

? Our use of \w, \d, \D, and \W is locale-sensitive: Sort::Naturally uses a "use locale" statement.

? When comparing two strings, where a numeric substring in one place is not up against a numeric substring in another, the non-numeric always comes first. This is fudged by reading pretending that the lack of a number substring has the value -1, like so:

```
foo       =>  "foo",  -1
foobar    =>  "foo",  -1,  "bar"
foo13     =>  "foo",  13,
foo13xyz  =>  "foo",  13,  "xyz"
```

That's so that "foo" will come before "foo13", which will come before "foobar".

? The start of a string is exceptional: leading non-\W (non-word, non-digit) components are are ignored, and numbers come before letters.

? I define "numeric substring" just as sequences matching m/\d+/ -- scientific notation, commas, decimals, etc., are not seen. If your data has thousands separators in numbers ("20,000 Leagues Under The Sea" or "20.000 lieues sous les mers"), consider stripping them before feeding them to "nsort" or "ncmp".

The nsort function

This function takes a list of strings, and returns a copy of the list, sorted.

This is what most people will want to use:

```
@stuff = nsort(...list...);
```

When nsort needs to compare non-numeric substrings, it uses Perl's "lc" function in scope of a <use locale>. And when nsort needs to lowercase things, it uses Perl's "lc" function in scope of a <use locale>. If you want nsort to use other functions instead, you can specify them in an arrayref as the first argument to nsort:

```
@stuff = nsort( [
                  \&string_comparator,   # optional
                  \&lowercaser_function  # optional
                ],
                ...list...
              );
```

If you want to specify a string comparator but no lowercaser, then the options list is "[\&comparator, '']" or "[\&comparator]". If you want to specify no string comparator but a lowercaser, then the options list is "['', \&lowercaser]".

Any comparator you specify is called as "$comparator->($left, $right)", and, like a normal Perl "cmp" replacement, must return -1, 0, or 1 depending on whether the left argument is stringwise less than, equal to, or greater than the right argument.

Any lowercaser function you specify is called as "$lowercased = $lowercaser->($original)". The routine must not modify its $_[0].

The ncmp function

Often, when sorting non-string values like this:

```
@objects_sorted = sort { $a->tag cmp $b->tag } @objects;
```

...or even in a Schwartzian transform, like this:

```
  @strings =
    map $_->[0]
    sort { $a->[1] cmp $b->[1] }
    map { [$_, make_a_sort_key_from($_) ]
    @_
  ;
```

...you wight want something that replaces not "sort", but "cmp". That's what Sort::Naturally's "ncmp" function is for. Call it with the syntax "ncmp($left,$right)" instead of "$left cmp $right", but otherwise it's a fine replacement:

```
  @objects_sorted = sort { ncmp($a->tag,$b->tag) } @objects;
```

```
  @strings =
    map $_->[0]
    sort { ncmp($a->[1], $b->[1]) }
    map { [$_, make_a_sort_key_from($_) ]
    @_
  ;
```

Just as with "nsort" can take different a string-comparator and/or lowercaser, you can do the same with "ncmp", by passing an arrayref as the first argument:

```
 ncmp( [
        \&string_comparator,   # optional
        \&lowercaser_function  # optional
       ],
       $left, $right
     )
```

You might get string comparators from Sort::ArbBiLex.

NOTES

? This module is not a substitute for Sort::Versions! If you just need proper version sorting, use that!

? If you need something that works sort of like this module's functions, but not quite the same, consider scouting thru this module's source code, and adapting what you see. Besides the functions that actually compile in this module, after the POD, there's several alternate attempts of mine at natural sorting routines, which are not compiled as part of the module, but which you might find useful. They should all be working implementations of slightly different algorithms (all of them based on Martin Pool's "nsort") which I eventually discarded in favor of my algorithm. If you are having to naturally-sort very large data sets, and sorting is getting ridiculously slow, you might consider trying one of those discarded functions -- I have a feeling they might be faster on large data sets. Benchmark them on your data and see. (Unless you need the speed, don't bother. Hint: substitute "sort" for "nsort" in your code, and unless your program speeds up drastically, it's not the sorting that's slowing things down. But if it is "nsort" that's slowing things down, consider just:

```
if(@set >= SOME_VERY_BIG_NUMBER) {
  no locale; # vroom vroom
  @sorted = sort(@set);  # feh, good enough
} elsif(@set >= SOME_BIG_NUMBER) {
  use locale;
  @sorted = sort(@set);  # feh, good enough
} else {
  # but keep it pretty for normal cases
  @sorted = nsort(@set);
}
```

? If you do adapt the routines in this module, email me; I'd just be interested in hearing about it.

? Thanks to the EFNet #perl people for encouraging this module, especially magister and a-mused.

COPYRIGHT AND DISCLAIMER

AUTHOR

Sean M. Burke "sburke@cpan.org"