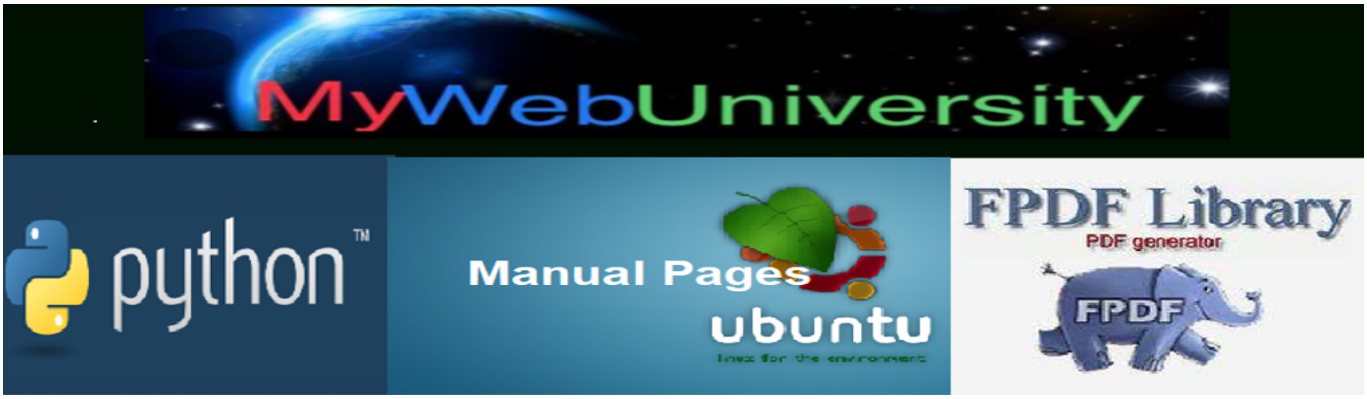

Rocky Enterprise Linux 9.2 Manual Pages on command 'perlhacktut.1'

$ man perlhacktut.1

NAME

perlhacktut - Walk through the creation of a simple C code patch

DESCRIPTION

This document takes you through a simple patch example.

If you haven't read perlhack yet, go do that first! You might also want to read through perlsource too.

Once you're done here, check out perlhacktips next.

EXAMPLE OF A SIMPLE PATCH

Let's take a simple patch from start to finish.

Here's something Larry suggested: if a "U" is the first active format during a "pack", (for example, "pack "U3C8", @stuff") then the resulting string should be treated as UTF-8 encoded.

If you are working with a git clone of the Perl repository, you will want to create a branch for your changes. This will make creating a proper patch much simpler. See the perlgit for details on how to do this.

Writing the patch

How do we prepare to fix this up? First we locate the code in question - the "pack" happens at runtime, so it's going to be in one of the pp files. Sure enough, "pp_pack" is in pp.c. Since we're going to be altering this file, let's copy it to pp.c~.

[Well, it was in pp.c when this tutorial was written. It has now been split off with "pp_unpack" to its own file, pp_pack.c]

Now let's look over "pp_pack": we take a pattern into "pat", and then loop over the pattern, taking each format character in turn into "datum_type". Then for each possible format character, we swallow up the other arguments in the pattern (a field width, an asterisk, and so on) and convert the next chunk input into the specified format, adding it onto the output SV "cat".

How do we know if the "U" is the first format in the "pat"? Well, if we have a pointer to the start of "pat" then, if we see a "U" we can test whether we're still at the start of the string. So, here's where "pat" is set up:

```
    STRLEN fromlen;
    char *pat = SvPVx(*++MARK, fromlen);
    char *patend = pat + fromlen;
    I32 len;
    I32 datumtype;
    SV *fromstr;
```

We'll have another string pointer in there:

```
    STRLEN fromlen;
    char *pat = SvPVx(*++MARK, fromlen);
    char *patend = pat + fromlen;
+   char *patcopy;
    I32 len;
```

```
    I32 datumtype;
    SV *fromstr;
```

And just before we start the loop, we'll set "patcopy" to be the start of "pat":

```
    items = SP - MARK;
    MARK++;
    SvPVCLEAR(cat);
+   patcopy = pat;
    while (pat < patend) {
```

Now if we see a "U" which was at the start of the string, we turn on the "UTF8" flag for the output SV, "cat":

```
+   if (datumtype == 'U' && pat==patcopy+1)
+       SvUTF8_on(cat);
    if (datumtype == '#') {
        while (pat < patend && *pat != '\n')
            pat++;
```

Remember that it has to be "patcopy+1" because the first character of the string is the "U" which has been swallowed into "datumtype!"

Oops, we forgot one thing: what if there are spaces at the start of the pattern? "pack("U*", @stuff)" will have "U" as the first active character, even though it's not the first thing in the pattern. In this case, we have to advance "patcopy" along with "pat" when we see spaces:

```
    if (isSPACE(datumtype))
        continue;
```

needs to become

```
    if (isSPACE(datumtype)) {
        patcopy++;
        continue;
    }
```

OK. That's the C part done. Now we must do two additional things before this patch is ready to go: we've changed the behaviour of Perl, and so we must document that change. We must also provide some more regression tests to make sure our patch works and doesn't create a bug somewhere else along the line.

Testing the patch

The regression tests for each operator live in t/op/, and so we make a copy of t/op/pack.t to t/op/pack.t~. Now we can add our tests to the end. First, we'll test that the "U" does indeed create Unicode strings.

t/op/pack.t has a sensible ok() function, but if it didn't we could use the one from t/test.pl.

```
 require './test.pl';
 plan( tests => 159 );
```

so instead of this:

```
 print 'not ' unless "1.20.300.4000" eq sprintf "%vd",
                                    pack("U*",1,20,300,4000);
 print "ok $test\n"; $test++;
```

we can write the more sensible (see Test::More for a full explanation of is() and other testing functions).

```
 is( "1.20.300.4000", sprintf "%vd", pack("U*",1,20,300,4000),
                                "U* produces Unicode" );
```

Now we'll test that we got that space-at-the-beginning business right:

```
is( "1.20.300.4000", sprintf "%vd", pack("  U*",1,20,300,4000),
                                    "  with spaces at the beginning" );
```

And finally we'll test that we don't make Unicode strings if "U" is not the first active format:

```
isnt( v1.20.300.4000, sprintf "%vd", pack("C0U*",1,20,300,4000),
                                     "U* not first isn't Unicode" );
```

Mustn't forget to change the number of tests which appears at the top, or else the automated tester will get confused. This will either look like this:

```
print "1..156\n";
```

or this:

```
plan( tests => 156 );
```

We now compile up Perl, and run it through the test suite. Our new tests pass, hooray!

Documenting the patch

Finally, the documentation. The job is never done until the paperwork is over, so let's describe the change we've just made. The relevant place is pod/perlfunc.pod; again, we make a copy, and then we'll insert this text in the description of "pack":

```
=item *

If the pattern begins with a C<U>, the resulting string will be treated
as UTF-8-encoded Unicode. You can force UTF-8 encoding on in a string
with an initial C<U0>, and the bytes that follow will be interpreted as
Unicode characters. If you don't want this to happen, you can begin
```

your pattern with C<C0> (or anything else) to force Perl not to UTF-8 encode your string, and then follow this with a C<U*> somewhere in your pattern.

Submit

See perlhack for details on how to submit this patch.

AUTHOR

This document was originally written by Nathan Torkington, and is maintained by the perl5-porters mailing list.