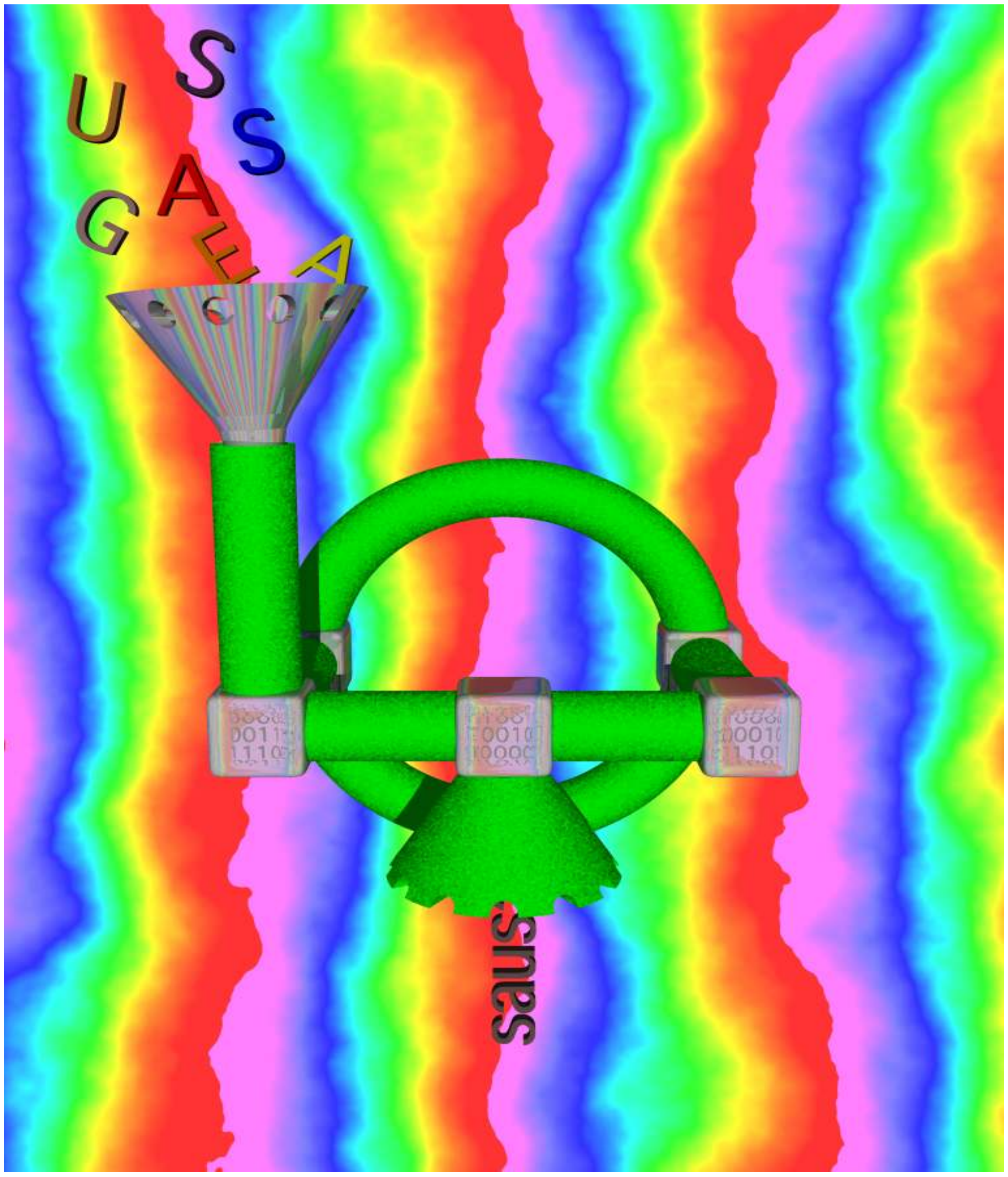
Perls Before Swine



January 11, 2012

Introduction

What is Perl?

Perl is a twenty-year-old scripting language designed for managing text. It is cross-platform, running on Linux, Unix, Mac OS X, Windows, and probably many more operating systems. It comes pre-installed on most operating systems today, and is used for managing server tasks, formatting documents, and filtering data. It may be the most-used programming language on the web, and has sometimes been called the duct-tape of the Internet.

If you're familiar with the use of duct tape, you'll have an idea of what Perl is used for. Perl is not the prettiest of solutions. But it works. It holds together things that would otherwise never hold together, and is a useful tool for creating quick solutions to thorny problems. There is an elegance in duct tape, an elegance in the solutions of the trenches. When something is broken it needs to be fixed.

The web site

You can find the latest version of this tutorial, as well as the resources archive, at http://www.hoboes.com/NetLife/Swine/.

Why use Perl?

If you're a web page designer and you're interested in programming, for example, you're probably already using PHP. What makes Perl useful instead of PHP? The answer is the command line. Perl excels as filter and as glue. It is great at taking some input—usually text—and modifying it. It acts as a great text sausage machine, grinding up text and spitting it out.

Perl also makes for a great glue tying together the various command line programs you use and automating your use of them. Perl scripts are often used as cron jobs, running automatically at specified times. Perl is a great way of taking what you want to give your command line program and converting it into what the command line program expects. It is great at mediating between two or more data sources.

If you manage a web site or a MySQL database and need regular backups and monitoring, or if you need to regularly collect and collate data from a set of files, Perl is a great tool to know. If your task is a series of changes—if you can think of it as a series of sieves or as an assembly line of tasks—Perl can provide rapid automation for that task.

What do you need?

Sample Data

Go to http://www.hoboes.com/NetLife/Swine/ to download the resources archive. Inside, you'll find a text file called "songs.txt". We'll be using that in this tutorial.

Text Editor

You will need a text editor, such as Smultron on the Macintosh or NoteTabPro on Windows. If you intend to edit Perl scripts directly on a remote server, you will need familiarity with a Unix text editor such as vi or pico.

If you're using a GUI text editor, you'll want to make sure that it saves your files with Unix line endings. It usually won't matter, but it can sometimes help track down errors.

Terminal or Shell

You will need to be able to execute your scripts. Normally you will do this from some sort of terminal or shell application. If you are running these scripts on a remote server, you will probably use ssh, or secure shell to get to that server. If you are running them on your local Mac OS X workstation, you'll use the Terminal application in your Utilities folder.

The basic Perl filter

Make sure you've downloaded the sample data, and then create the following text file:

```
#!/usr/bin/perl
while (<>) {
	print;
}
```

Save this file as the filename *show*. Once you've saved it, make sure that it is *executable* by you. Go to your command line, make sure that your are in the correct folder, and type:

```
chmod u+x show
```

On Mac OS X, you can ensure that you are in the correct folder by typing "cd" in the terminal, a space, and then dragging the folder onto your terminal window. Press return, and you will *c*hange *d*irectory into that folder.

Now, type:

```
./show show
```

The script should show you itself. Make sure that songs.txt is in the same directory as your script, and type:

```
./show songs.txt
```

The show script should show you all 7,006 lines in the songs.txt file.

What is it doing?

This is about as simple of a Perl script as you can get. While it doesn't do much yet, this is a shell around which you can build quite a few useful scripts.

```
#!/usr/bin/perl
```

The first line is not Perl. The first line tells the operating system what language this script is written in. More specifically, it tells the operating system which program can *interpret* this script.

Most shell scripts use the pound character ("#") for *comments*. What it really means is that every line that begins with a pound character is ignored by the scripting language. So Perl ignores this line because it begins with a pound character, but the operating system or shell that you're using knows to send this script off to the program called /usr/bin/perl.

If your computer didn't come with Perl pre-installed, you may have it installed in /usr/local/bin/perl instead. Nowadays, however, most operating systems come with Perl pre-installed.

```
while (<>) {
}
```