

Perl as Glue



Perl makes excellent **glue code**
as it allows you to easily string unrelated commands
and programs together into one complete package

Perl Origins

Perl has its origins in **system administration reporting**, and **text processing**

It draws inspiration from these classic Unix tools:

- * Bourne shell
- * awk
- * sed
- * grep
- * tr

Regexes

We can either use Perl's built-in versions of these tools, or treat Perl like a shell-scripting language and tie these together through the backtick quote operator (``` or `qx{}`)

Raiding the Greybeard

Use Perl's built-ins

- # Portable code
- # Better performance
- # Better control over errors

Use the shell's tools

- # R.A.D. to the max
- # Greybeard-friendly

My recommendation:

Where possible you should use Perl's built-in functions

However, we'll do a little bit of both today

Perl Variables

Here are some variables you should know about:

```
|||_ \
(_-< /_/
/_/(_)
||
```

Sub-process exit status

Set by backticks and system()

```
||_| |
(_-<|_|
/_/(_)
||
```

Status of system call; errno

Set when you run a built-in function

```
^
||| ^ | /_ \
(_-< | ( ) |
/_/ | \_/_/
||
```

Name of the current OS

Similar to output of "uname" program

Redirection

The shell's redirection operators control where a command's output goes

And where its input comes from

Today you may see these variations:

Syntax

`cmd > /dev/null`

`cmd 2> /dev/null`

`cmd 2>&1`

`cmd1 | cmd2`

Meaning

Send STDOUT to the null device
Print just the error messages

Send STDERR to the null device
Only print normal output

Combine STDERR with STDOUT

Make cmd1's STDOUT become cmd2's STDIN



Of course, Perl has the usual compliment of text processing functions

classics such as:

- * substr()
- * index()
- * rindex()

but the match operators are almost always what you want