Coding practices and conventions

Ways to understand each other's code

Luis Fernando Muñoz Mejías

Universidad Autónoma de Madrid

6th Quattor Workshop
Nikhef, 10/2008

Outline

Outline

Does it? Really?

- We all make mistakes
- We all have bad habits:

```
system(“my command $arg1 $arg2”);
open(FH, “>$filename”);
open (PIPE, “|$command with $arguments”);
```

Outline

Many eyes

Quoting a talk on Linux Kernel style:

- I want to read your code
- I want you to read my code
- I want to fix your bugs
- I want you to fix my bugs

User's mistakes

- Users make mistakes on their configuration
- We must provide them with useful, consistent output to fix their own mistakes
- We are inconsistent
  - Some modules are too noisy
  - Some modules are too quiet

Outline

Our code runs with root privileges!!

- Bad habits lead to security issues
- From ncm-nfs

```
$mnt = $hashref->{mntpt};
system("umount -l $mntpt");
```

  Do not trust the profile's contents!!
- The tainted mode won't catch everything

Outline

Indentation

- During the 4th workshop we agreed to use 4 whitespaces, and no tabs
  - Not my favourite decision, but it's the official one
  - Fix the code you modify
  - Fix your editor's configuration

White spaces

- Use spaces after keywords, such as `if` or `while`

```
if (foo) {
while ($bar) {
```

- Use whitespaces on funcion arguments
  - Between the function name and the list of arguments
  - After commas
  - **Not** before commas
  - **Not** before a closing parenthesis

```
foo (bar, baz)
```

- It may be OK not to use whitespaces before parenthesis when there are no arguments

```
foo()
```

Parenthesis

- Be generous
- If in doubt, use them
- If you have a function call with no arguments, use paranthesis anyways
  - `foo()` is easier to understand than `foo`

Line length

- No lines longer than 80 characters
- If your code goes beyond those 80 characters, fix it
- If you are printing longer strings, split them

```
print “There was Eru, the One, who in Arda is called
print join (“ “, qw (There was Eru, the One,
    who in Arda is called Ilúvatar;
    and he made first the Ainur,
    the Holy Ones, that were the
    offspring of his thought,
    and they were with him before
    aught else was made));
```

Modularity

- It matters!!!
- Who can read a `Configure()` method that is 500 lines long?
- Who remembers a variable that was declared 300 lines ago?
- If your block goes beyond the screen on either direction, fix it
- If you need more than 9 local variables, split your code
- Functions should do just one thing, and do it right

Comments

- Comment Pan data structures on the Pan code
- Comment function headers
  - Comment only what they do
- Don't comment function bodies
  - Except very tricky, special parts
  - Don't comment bad code, re-write it
- No credits in your comments, except on file headers
  - Old AII is a great example of what shouldn't be done ;)
- Keep your comments and code consistent!!

Prefer methods over plain functions

- When writing a component, we want to log unexpected things
- Have `$self` and let it log

```
sub foo {
    my $self = shift;
    $self->info ("Hello, world!");
```

- Use normal functions only if you need to export them

Program for Perl 5.8 or later

- Don't `use vars`
  - It was obsolete on SL3
  - For package-wide variables, `our` is the preferred way

```
our @ISA
our $ec =
```

Outline

Don't use `system()` or qx

- Dont use `LC::Process` directly either!!
- Use `CAF::Process`, as it logs the command being run
- The command's exit status is stored in `$?`

CAF::Process examples

- Running a command

```
my $proc = CAF::Process->new ([qw (touch /foo)],
    log => $self);
$proc->run ();
```

- Getting a command's output

```
my $proc = CAF::Process->new (["ls", "-lh"],
    log => $self);
my $ls = $proc->output ();
```

- Piping to a command

```
my $cmdin = "Some text to send in";
my $proc = CAF::Process->new (["cat", "/dev/fd0"],
    log => $self, stdin => $cmdin);
$proc->execute ();
```

Don't open for writing

- Use `CAF::FileWriter`
- It will not blindly overwrite the file
- It will prevent symlink attacks
- It can cancel your printed contents, keeping the original file unchanged

```
my $fh = CAF::FileWriter->open (“/some/file”,
    log => $self);
print $fh “There was Eru, the One...”;
if ($error) {
   $fh->cancel ();
}
$fh->close ();
```

Don't use temporary files

- Use a `CAF::FileWriter` object and cancel or use an in-memory file
- If you need to feed a command, use a pipe
  - See the `CAF::Process` examples

Miscelaneous file handling

- Prefer `LC::File` over `File::Copy`, `File::Path` and friends
- It won't follow symlinks
- Your files will end where you want them to

Further improvements to the CAF library

- What tasks do we repeat over and over?
- Do we need a `CAF::FileReader`?
- Should we re-write and improve `NCM::Check::lines`?
- A module to download and verify files from the Internet?
- What else?

Outline

Make your code run in tainted mode

- Don't trust any inputs
- Not even from the profile
- Sanitize all values you'll pass as system calls
  - Command arguments
  - File paths
  - Sanitize everything just after reading it!!
- You can be (slightly) more relaxed when printing to a file
  - Arguments to print are not checked for taintedness

Printing Bash scripts

- Print values between single quotes
- Check the value doesn't have its own single quotes
  - Perhaps a `shellQuote()` method somewhere?
- Do this for everything you write on /etc/sysconfig!

Outline

Don't annoy the user

- Use `info` and `ok` only for **very** important stuff
- The console should be kept clean for everything but errors
  - Users may need to filter the output of child commands

But if the user asks to...

- Be generous with `verbose` messages
- Print user-relevant stuff
  - Services being enabled or disabled
  - DNS queries being performed
  - URLs being accessed
  - ...
  - Don't log commands or files being written
    - `CAF::Process` and `CAF::FileWriter` will do it for you
- Don't print developer-relevant messages with `verbose`

Use debug for developer-relevant stuff

- Evolution of a variable, temporary contents... are developer-relevant contents
- The user doesn't care about them
- Do we have/need a convention for debug levels?
  - We currently set level 5 for almost everything

Outline

Code quality is a real need

- It reduces bugs
- It makes other developers' job easier
- It avoids re-inventing the wheel over and over
- New developers will understand things faster
- Technical students will be more productive

Outline

What's next

- A wiki page with all the conventions we just agreed
- Links everywhere to this wiki
- Tools to enforce these conventions
  - We already have some
    - We check against `die` on NCM components
- A framework to make unit testing easier

More reading...


Perl style guide `http://perldoc.perl.org/perlstyle.html`


Greg Kroah-Hartman, Documentation/Coding style and beyond `http://www.kroah.com/linux/talks/ols_2002_kernel_codingstyle_talk/html/`


Building an OpenBSD port `http://www.openbsd.org/porting.html`