# 1 Templating

Templates are a generalized view of some data, that can be processed to generate the actual output. Templates can be simple, or complex. Generally, templates involve replacement text. For example, lets say we have a template below:

```
<html>
<head>
    <title><?=$title?></title>
</head>
<body>
   ...
</body>
</html>
```

Given some definition for `$title`, we can create the output that looks like HTML.

There are many templating engines out there. That being said, the best ones are the ones you come up with yourself. They're generally easy to write, and you can come up with one in a few minutes. Here's one I just wrote:

```
#!/usr/bin/perl

# simple templating script
# Alex S., http://www.theparticle.com/

my (%args);
map {($a,$b)=split/=/; $args{$a} = $b ? $b : 1} @ARGV;
my %_PARAM = %args;

{   local $/=undef; $_ = <>;
    1 while(s{<\?include(.*?)\?>}{
        $a=$1; $a=~s/^\s+['"]?|['"]?\s+$//sgi;
        $a=eval('"$a"');
        open my $in,$a or die "$a: $!"; $a=<$in>;
        $a=~s/^\s+|\s+$//sgi; $a}sgie);
}

$_ = '?>'.$_.'<?perl';
s{<\?=(.*?)\?>}{<?perl print $1; ?>}sgi;
s{\?>\s*(.*?)\s*<\?perl}{
    $a=$1; $a=~s/\'/\\'/sgi; "print '$a';"}sgie;

eval($_);
```

The way it works is: it reads standard input, along with arguments. The output goes to standard output. The arguments specify what to use for various values. For example, lets say we have `index.tpl`:

```
<html>
<head>
    <title><?=$_PARAM{title}?></title>
</head>
<body>
    <h1><?=$_PARAM{title}?></h1>
    <p>This is a paragraph.</p>
    <?perl
        for(1..10){
            ?>Hello World <?=$_?><br><?perl
        }
    ?>
</body>
</html>
```

We can run the templating script against this, ie:

```
$ cat index.tpl | perl tpl.pl - title='The Title!' >index.html
```

The output `index.html` would look like this:

```
<html>
<head>
    <title>The Title!</title>
</head>
<body>
    <h1>The Title!</h1>
    <p>This is a paragraph.</p>
    Hello World1<br>
    Hello World2<br>
    Hello World3<br>
    Hello World4<br>
    Hello World5<br>
    Hello World6<br>
    Hello World7<br>
    Hello World8<br>
    Hello World9<br>
    Hello World10<br>
</body>
</html>
```

You can get a bit more creative, and make a `header.inl` file:

```
<a href="<?=$_PARAM{link1}?>"><?=$_PARAM{text1}?></a> |
<a href="<?=$_PARAM{link2}?>"><?=$_PARAM{text2}?></a> |
<a href="<?=$_PARAM{link3}?>"><?=$_PARAM{text3}?></a>
```

You can then include it in the original template, as the header:

```
<html>
<head>
    <title><?=$_PARAM{title}?></title>
</head>
<body>
    <center><?include header.inl ?></center>
    <h1><?=$_PARAM{title}?></h1>
    <p>This is a paragraph.</p>
    <?perl
        for(1..10){
            ?>Hello World <?=$_?><br><?perl
        }
    ?>
</body>
</html>
```

Run the templates:

```
$ cat index.tpl | perl tpl.pl - title='The Title!'    \
      link1='http://www.google.com' text1='Google'    \
      link2='http://www.yahoo.com/' text2='Yahoo!'    \
      link3='http://www.theparticle.com/'             \
         text3='Teh Particle Revelation' > index.html
```

The output becomes:

```
<html>
<head>
    <title>The Title!</title>
</head>
<body>
    <center><a href="http://www.google.com">Google</a> |
<a href="http://www.yahoo.com/">Yahoo!</a> |
<a href="http://www.theparticle.com/">Teh Particle Revelation</a></center>
    <h1>The Title!</h1>
    <p>This is a paragraph.</p>
    Hello World1<br>
```

```
    Hello World2<br>
    Hello World3<br>
    Hello World4<br>
    Hello World5<br>
    Hello World6<br>
    Hello World7<br>
    Hello World8<br>
    Hello World9<br>
    Hello World10<br>
</body>
</html>
```

Neat, no? But your `header.inl` template has the three links. What if you wanted to include as many links as are needed? Well, that can easily be changed by modifying the `header.inl` itself:

```
<?perl
    for($i=1;$_PARAM{'link'.$i};$i++){
        if($i > 1){
            ?> | <?perl
        }
        ?><a href="<?=$_PARAM{'link'.$i}?>"><?=$_PARAM{'text'.$i}?></a><?perl
    }
?>
```

Now, it will include any number of links you specify, instead of the magic number '3'.

# 2 Other Data

Don't get the impression that this only works for HTML. You can use this for any purpose you like. Lets say you run a website, and would like to notify your users of something. You have a file `email.csv`, that has all users and their emails, ie:

```
John Jackson,john@msn.com
Jack Johnson,jack@yahoo.com
John Doe,doe@gmail.com
Jane Doe,jdoe@hotmail.com
```

Now you can create your remplate email, ie:

```
Dear <?=$_PARAM{name}?>,
    Your account is currently disabled.
Thank you.
```

Now, to send everyone a customized email, you can do:

```
cat emails.csv | while read line; do
    name=‘echo $line |cut -d, -f1‘;
    email=‘echo $line | cut -d, -f2‘;
    cat email.tpl | perl tpl.pl - name=$name | mail $email;
done
```

Of course, you'll need to do this on a unix system, that has properly configured `mail` command.

The possibilities are endless :-)