Anti-patterns and other mistakes to avoid

Multiple sources for files

BAD

This is a pattern that proliferated puppet 4 by copy/paste

```
      "puppet:///modules/foobar/${::mc_grid}/${::hostname}/...",
      "puppet:///modules/foobar/${::mc_grid}/...",
      'puppet:///modules/foobar/...',
```

Problem:

- Fileserver lookups are one of the slowest operations when compiling/applying a catalog and the puppet fileserver can only handle a limited number of concurrent requests. Each additional source means one extra API request to puppetserver and one extra disk read to check if the file exists.
  If you have an excessive and unnecessary number of sources that will slow down puppet runs for every host in the grid

Better patterns:

- EPP templates and hieradata
- ERB templates and hieradata
- Hardcode exceptions

```
class foobar::config {
  $public_key_source = $::mc_grid ? {
    'DEV'   => 'puppet:///modules/foobar/foobar.dev.pem',
    default => 'puppet:///modules/foobar/foobar.org.pem',
  }
  file { '.....':
    source => $public_key_source,
  }
}
```

FACTERLIB environment variable to load custom facts

TODO

GRID/Layer if-else spaghetti

BAD

```
if $::mc_grid == 'DEV'
  $user = 'bla'
elsif $::mc_grid =~ /US\w|ZA|DE/ and $::mc_servertype == 'abc'
  $user = 'foo'
else
  if $::mc_servertype == 'xzc'
    $user = 'bla'
  else
    $user = 'bar'
  fi
end
file { '...': user => $user }
```

GOOD
Move complex **data** logic into hieradata.

- Explicit default value
- Variations can be added without increasing code complexity

```
class foo(
  String $user = 'mcuser',
)
file { '...': user => $user }
```

```
$ grep 'foo::user' data/
data/grid/DEV.yaml: foo::user: 'bla'
data/grid/UK.yaml: foo::user: 'foo'
data/role/xyz.yaml: foo::user: 'bar'
```

Lookup() in class params

BAD

```
# This is bad.
class foo(
  String $user = lookup('foo:bar_user')
)
```

- **BAD: lookup() as the default value for a class param.** `Type <variable> = <default vaulue>` A `lookup()` function as the class param default value will only execute after the normal auto-lookup. This can cause many more unnecessary data lookups:
  - 1st. auto-lookup will search for `foo::user` in global hiera
  - 2nd. auto-lookup will search for `foo::user` in module hiera
  - 3rd. lookup() will search for `foo::bar_user` in global hiera
  - 4th. lookup() will search for `foo::bar_user` in module hiera

- **BAD: Doesn’t follow naming conventions.** Another common mistake is when `lookup()` doesn't follow class variable naming convention. Users expect `foo::user` to be the hiera variable setting `$user` . By not following `<class_name>::<variable_name>` convention you are reducing readability and supportability

GOOD

- Follow `<class_name>::<variable_name>` variable names in hiera
- Don’t use lookup() in class params
- If unconventional lookups are needed, put them in the class body

```
class foo(
  String $user,
) {
  $reposerver = lookup("reposervers.${::mc_grid}", String, 'first',
'repo.uk.mimecast.lan')
}
```

If/Else conditional spaghetti in templates (specially if json)

BAD

```
{
  "port": 8516,
  "usessl": true,
  "intragridssl": true,
    <%- if @mc_grid == 'USPCOM' -%>
     "keystorepath": "/usr/local/mimecast/mimecast.lan.bcfks",
    <%- else -%>
     "keystorepath": "/usr/local/mimecast/mimecast.lan_keystore",
    <%- end -%>
  ...
}
```

GOOD

```
# templates/foo.json.epp
{
  ..
  "keystorepath": "<%= $::foo::keystorepath %>",
  ...
}
```

```
class foo(
  String $keystorepath,
) ...
```

```
data/common.yaml: keystorepath: '/usr/local/mimecast/mimecast.
lan_keystore'
data/grid/USPCOM.yaml: keystorepath: '/usr/local/mimecast/mimecast.lan.
bcfks'
```

Inconsistent variable names and namespacing

BAD

```
# hieradata/...
---
bar_webhook: "https://.." # Where is this used? I need to grep -R all
modules to find out
foo::config::timeout 5    # Too generic, timeout for what?
```

```
# scripts.pp
class foo::config(
  String $baraddress = lookup("bar_webhook") # Where is $baraddress
used? I have to do a grep -R to find out
  Integer $timeout = lookup("foo::config::timeout"),
) {
  contain foo::scripts::barfoo
  file { 'localproperties.json': content => epp('foo/localproperties.
json.epp') }
}
```

```
# scripts/barfoo.pp
class foo::scripts::barfoo {
  $server_address = $foo::config::baraddress
  file { 'script.sh': content => epp('foo/bar.sh.epp') }
}
```

```
#!/bin/bash
server=<%= $server_address %> # I have no clue how or where
server_address was set.
```

```
{
  "rbl_read_cache_timeout" => <%= $timeout %> # I have no clue where
this is set
}
```

- All the variable names are inconsistent.
- Variable namespacing is not respected in hiera.
- Looking at hiera you have no clue where/why/how that value is used
- Looking at the script, you have to follow multiple dependency links until you find how the value is set

GOOD

```
# hieradata/....
foo::scripts::barfoo::server_address: "https://..."     # I know
exactly where this is used.
foo::config::localproperties_rbl_read_cache_timeout: 5  # I know
exactly the module, file and setting this is configuring
```

```
# scripts.pp
class foo::config(
  Integer $localproperties_rbl_read_cache_timeout, # Auto-lookup from
hiera
...
```

```
# config/barfoo.pp
class foo::scripts::barfoo(
  String $server_address, # Auto-lookup from hiera
...
```

```
#!/bin/bash
server=<%= $foo::scripts::barfoo::server_address %> # I can guess this
is in hiera with the exact same name
```

```
{
  "rbl_read_cache_timeout" => <%= $foo::config::
localproperties_rbl_read_cache_timeout %> # I can guess this is in
hiera with the exact same name
}
```

- **Respect variable name and namespacing.**
  - <module_name>::<namespace_where_the>::<variable_is_needed>::<variable_name_that_is_descriptive_and_matches_how_and_where_its_used>No guessing needed!!!
  - `$foo::config::localproperties_rbl_read_cache_timeout`
    - module foo
    - class foo::config
    - used to configure `rbl_read_cache_timeout` in the `localproperties` config file
- Avoid `lookup()` functions. Prefer auto-lookups.