À faire/à pas faire

ALBERT SHIH

Direction Informatique
Observatoire de Paris

13 avril 2023

Syntaxes

- Utilisez vérificateur de syntaxe
- Intégration dans neovim/vim/emacs (langage server)
- Vérification en cli

  ```
  puppet-lint apache.pp
  ```

- Possibilité de correction avec

  ```
  puppet-lint --fix apache.pp
  ```

- Atttention bien «vérifier»
- Utilisez : vim/neovim/emacs/etc. avec une validation & snippets

Syntaxes

- Attention aux quotes ’ $\neq$ ",
- On utilise " pour l’interpolation
- `class { "La classe ${machin}" : }`
- On ne met pas de pour une string non interpoler.
- Interpolation : Ne pas oublier les { }.
- Respecter les indentations.
- Respecter les espaces
- Pas de tab mais des «espaces»
- Aligner les =>

Variables

- Éviter les variables globales `$machin` ... sauf raison intrinsèque & claire `$bash_path`, `$dio_softwares_sbin_path`, etc.
- Utilisez de préférence les variables d'une classe `$dio_truc::machin`
- Utilisez de préférence des noms de variables explicites pour ceux utiliser ailleurs.
- Utilisez de préférence des noms de variables qui commence par _ pour les variables locales.
- Typez les paramètres des classes/defines.
- Chronophage

Variables

- Accès à valeurs

```
include $dio_truc
file {'mon fichier':
  ensure  => 'present',
  path    => '/qcqpart',
  content => "La valeur est ${dio_truc::machin}",
}
```

- Utilisation

```
class dio_scripts::check_znapzend ()
{
  include ::dio_monitoring
  $script_path = "${::dio_monitoring::plugins_contrib_path}/check_znapzend"
}
```

Variables

- Utilisation

```
class dio_scripts::ldap2alias ( String $script_name = 'ldap2alias',)
{
  $dio_softwares_sbin_path = lookup('dio_softwares_sbin_path')
  $script_path = "${dio_softwares_sbin_path}/${script_name}"
}
```

```
class dio_smtp::aliases {
  require ::dio_auth_ldap
  class { 'dio_scripts::ldap2alias':
    ldap_host        => $::dio_auth_ldap::first_ldap_host,
    ldap_bind_dn     => $::dio_auth_ldap::dsa_createaliases::dsa,
    ldap_bind_passwd => $::dio_auth_ldap::dsa_createaliases::passwd,
    ldap_base        => $::dio_auth_ldap::ldap_base,
  }
  $create_aliases_script = $::dio_scripts::ldap2alias::script_path
}
```

Régle de platine

- Ne pas dupliquer une information.
- Vrai pour tout
  - Information
  - module (éviter le `cp toto.pp tata.pp` en particulier dans `dio_host`)

Les autres régles (fichiers)

- Éviter `file` pour modifier un fichier
- Préférence : `augeas`, `file_line`
- . . .sauf si.
- Éviter de bloquer une configuration d'un objet à un endroit.

Les autres régles (fichiers)

- 4 méthodes :
  - `file` (possibilité recursif)
  - `file` + `template`
    - `erb`
    - `epp`
  - `concat`
- `epp > erb`
- `file + template > file`
- Si possible rajouter commentaire

Les autres régles

- Éviter `custom_fragment`.
- Préférence : `apache::mod::`,
- . . .sauf si impossible.

Les autres régles

- `package` vs `ensure_packages`
- Si possible : `ensure_packages` > `package`

yaml vs module

- Là où sont les données
- Avantages
  - Dossier ’data’ dans chaque module.
  - Chargement définit par le fichier ‘hiera.yaml‘
  - Remplace les `if $facts['os']['family']`
  - Possibilité de faire des traitements ;
- Desavantages
  - peut devenir illisible.
  - peut devenir mystérieux.
  - Ne fonctionne pas toujours.

yaml vs module

- ben . . .ça dépend.
- quand possible : `yaml`, sinon . . .
- Lisibilité.

Modules

- Théorie
- Un morceau de code indépendant.
  - Pas de données non lier au but
  - Possibilité de mettre publique
- Ensemble modules + profiles + roles + data
- Pratiquement
- Prefix par `dio_`
  - Minimum de données non lier au but

Profiles/Roles

- Origine : Palier à l'absence de héritage
- Profiles : Regroupe un ensemble d'opérations
- Rôles : Regroupes un ensemble de profiles
- Une machine/VM = un ou des rôles.
- Théoriquement les profiles utilisables sans comprendre.
- Théoriquement paramètres uniquement pour les roles

Profiles/Roles

- Pratiquement

```
[jas@io data][master]$ grep -E  'dio_profiles::.*:' *.yaml |wc
    873    1599   60024
[jas@io data][master]$
```

Exemple

```
class dio_roles::indico {
  include dio_profiles::client_sso
  include dio_profiles::nginx
  include dio_profiles::postgresql
  include dio_profiles::redis
  include dio_profiles::indico
}
```

```
class dio_profiles::nginx ( Boolean $manage_repo = false) {
  class { '::nginx':
    manage_repo   => $manage_repo,
  }
  include ::dio_logrotate
  $nginx_pid = $::nginx::pid
  $nginx_logdir = $::nginx::log_dir
  dio_logrotate::rule { 'nginx':
    path          => "${nginx_logdir}/*.log",
    postrotate    => "[ ! -f ${nginx_pid} ] || kill -USR1 `cat ${nginx_pid} `"
  }
  dio_monitoring::add_procs { 'nginx':
    description => 'Service nginx',
    critical    => '1:',
  }
}
```

Chargement class

- `include`
  - obligation yaml
  - peut être multiple
- `require`
  - Idem que ‘include‘ mais avec dépendance
  - risque de boucle
- `class`
  - Passage paramètres
  - unique.

Profiles vs modules

- Compartiment vs Lisibilité vs taille

Les autres régles (module)

- Regarder si un module existe sur la forge.
- Ne pas recopier un module maison, mais l'améliorer.
- Ne faites pas dans un module

```
if $hostname ==
```

La règle de diamant

puppet est là pour nous simplifier la vie. . .dans le temps