Puppet content management using GIT

Version tracking:

0.1	Cspence	18/6/2012	Initial draft
0.2	Cspence	19/6/2012	Comments
0.3	Cspence	20/6/2012	Further edits
1.0	Cspence	21/6/2012	Version 1.0
1.1	cspence	14/12/2012	Version 1.1

Contents

Introduction

The basic technique to managing a Puppet code content development cycle is to ensure that the process is well communicated and understood by all, with basic requirements for

1 – All content (code and optionally data) must be managed through version control

2 – Content should only be edited through a change process (not covered by this document)

3 – Content must go through testing before being deployed to a live environment (incidents or outages may not be subject to this stricture)

Example Puppet Environments:

Environment	Git branch	Filesystem path
Puppet-DEV	puppet-dev	/etc/puppetlabs/puppet/env/puppet-dev
Dev	dev	/etc/puppetlabs/puppet/env/dev
QA	qa	/etc/puppetlabs/puppet/env/qa
UAT	uat	/etc/puppetlabs/puppet/env/uat
Production/DR	production	/etc/puppetlabs/puppet/env/production

Here you can see the mapping between Puppet, the git repository and the filesystem. The Puppet environment maps to a specific development branch in the Puppet code git repository, and hence nodes can be pinned or switched between different versions of code at will.

The configuration of which environment a node should point at is controlled by the node's puppet.conf file

```
[root@puppet env]# grep environment /etc/puppetlabs/puppet/puppet.conf
    environment = production
```

By default a Puppet installation will configure the node to be in the 'production' environment. This needs to be configured on a per node basis if a node needs to be configured to get content from an alternative environment.

The modulepath and manifest directories per environment can be dynamically configured to reflect the client environment setting (either at the command line using `--environment $environment` or in the puppet.conf configuration file). The configuration format for this, which should go in the `[main]` portion of the configuration file on the Puppet master, is:

```
    modulepath = /etc/puppetlabs/puppet/env/$environment/modules:/opt/puppet/share/puppet/
modules

    manifest = /etc/puppetlabs/puppet/env/$environment/manifests/site.pp
```

Management of the client side environment could be carried out with:

- A custom fact to implementation to dictate which environment the node is in which would feed into a module to manage puppet.conf and write the correct environment for the client in the catalog (optional) - this could be derived from the hostname (node.environment.fqdn.moo.split('.')[1]) or static data, such as an /etc/role file managed out of band.
- A module to manage the puppet.conf using custom fact to populate the environement configuration stanza in the puppet.conf config file (along with a reference to the correct Puppet master) using fact data
- Use of the `--environment ${environment}` switch on the command line

Puppet masters

There may be multiple Puppet masters used as a way to distribute load, provide high availability and segregate client nodes. Each Puppet master will still have a copy of the code as represented by the git repo although that code might be pre-production. The code might be pushed or pulled automatically to/by the puppet master in question or the process might be driven by a human manually.

It is suggested that on the dev/test Puppet master each developer of *Puppet code* be given their own environment to develop against probably using a dynamic environment methodology using post-receive hooks to create the appropriate filesystems. In this way it is trivial to know where to check out the repository to in order to develop content in isolation (still using the branching methodology outlined in this document however). Using dynamic environments based on git branches is a solution to this.

Tracking content within environments

Despite the fact that we are tagging specific commits and merges to the repository to mark points in time with meaningful labels, it is important to track which version of the repository should be currently checked out to an environment. An alternative to this is to always regard the latest commit in a branch as the authoritative one. In this case, the trigger for a change is a branch merge and a git pull on the master. Using commits, a further data source and application will be required to pin each branch to the correct commit.

This is a sample table that could be used to track this.

Puppet master	Puppet environment	Environment path	Tag	Date
puppet	puppet-dev	/etc/puppetlabs/puppet/env/puppet-dev	Branch – Puppet-DEV	20/6/2012
puppet	production	/etc/puppetlabs/puppet/env/production	Tag – 2012062000	20/6/2012

A wiki, or other support resource would be a good place to store this data – it should be available to all Puppet code developers.

Using Puppet to manage repository/environment mappings

The Puppetlabs vcsrepo type (the current version at the time of writing 0.1.1) could be used by the puppetmaster to update code using a `vcsrepo` resource.

For this approach to work with specific tags, an external data source would be needed (be that Hiera or an ENC such as dashboard) to determine which revisions/tags/branches apply to a particular Puppet environment.

We recommend for deployment adopting a manual controlled solution until familiarity with the process is gained, and then move to an automated solution once comfort and trust in a solution is reached.

Notes

Development roles

It is important that great care be taken over merging code from topic branches into branches that are actually checked out to a Puppet environment and serve code to nodes. One or more 'Merge Masters' may tasked with performing merges from topic branches into code branches to retain control of this process. This approach has several advantages around familiarity with driving the git application to best effect, but can also be used as an effective style and code guideline gateway to hopefully avoid merging badly written or poorly formatted code (or code that doesn't fit whatever guidelines your organisation adopts) into a production environment. Merge masters should communicate effectively with each other to ensure consistency of approach and keep a log of updates.

Those tasked with simply checking out code should not be given write access to the repository to avoid having effectively unmanaged code merged back into the repository.

Automated testing

There are 2 basic tests that should at minimum apply to all code before it is merged into a production branch (and preferably even before then). The first is to make sure that it passes syntax validation (using Jenkins or other CI environment would facilitate this)

Syntax checking can be achieved by using Puppet itself:

```
find . -type f -name *.pp -exec /usr/local/bin/puppet parser validate '{}' \;
```

Style checking is also useful from a consistency point of view - it is well worth having code guidelines so that every member of a team has expectations as to quality, and you are going to be able to easily consume other team members code:

```
find . -type f -name *.pp -exec /usr/local/bin/puppet-lint '{}' \;
```

Using rspec and rspec-puppet (outside the scope of this document) to further automate testing is recommended to ensure basic unit testing of module and ruby extension code to validate expected behaviour of the code and the catalog.

Branches

Changes to Puppet content should start out life as a branch from the development environment which represents the bleeding edge of the production code, then be merged into the UAT environment (in this example). The content might be tagged for ease of reference at this time (it's easier to reference a tag than a commit checksum since tags probably follow some convention for tag naming). A lesser or greater number of branches may be appropriate, or you may even choose to have just one set of production code and manage changes through changes to data that the Puppet code consumes.

Once the content is regarded as tested, the content should be merged into the UAT environment (or whatever the next branch towards production is).

Again, the content should be tested and once satisfaction as to quality is reached, it should be merged into the Master branch. Once merged into the master branch, the content should be tagged (which is a point in time reference to a particular commit – i.e. the one where you merged your production ready code into the Master branch) and that tag should be checked out into the appropriate environment.

This way, the most up to date environment is generally the development environment – newer changes get merged into this branch first, so it should be the one that is branched from in order to implement a change, or new feature.

Commit messages:

These are useful – the temptation is always to enter a random string of characters to get through the hurdle. Resist the temptation and use proper commit messages. If you succumb to the temptation, go back later review the messages and squash them before writing out to the repository.

Git accounts

Every user should have their own git account – this heightens the ability to track code and changes in ways that having shared accounts just doesn't facilitate.

Role based access should be implemented, preferably at the branch level, to avoid having unmanaged code merged back into the repository. git blame is your friend.

Environment access control

Control over what code should be deployed to a particular Puppet environment, assuming different teams need to check out code to different environments can be controlled using file system permissions assigned to different groups and ability to log into a different Puppet master.

This requires at least one different user account for each role (i.e. a user that has permissions to deploy to the production environment on the production Puppet master, a user that has permissions to deploy to the UAT environment on the UAT/QA Puppet master etc etc), or for individual user accounts belonging to named individuals and group membership, and using those groups on the directory permissions appertaining to a particular environment.

Those tasked with merely checking out code should not be able to write back to the repository. Thus for example your Puppet master should not be able to write back to the repo.

Code management process:

Code structure

The repository structure might be as follows:

```
Projects -> Puppet -> PuppetCode
       |-- manifests
       |   `-- site.pp
       `-- modules
          |-- ntp
          |   `-- manifests
          |       `-- init.pp
          `-- apache
              `-- manifests
                  `-- init.pp
```

Breaking out the manifests and modules into separate repositories makes life less manageable, though it does mean that changes to node definitions are subject to the same change control and process as code. Even if the decision is taken to manage the manifests directory and modules directory as different repositories, the same change control should apply to both. If you're using an ENC, you're probably only setting top level resource defaults and a default node declaration in the site.pp anyway, so it shouldn't change too much and thus having it in the git repo makes sense.

Getting content into Puppet-dev

The entry point to the process should be a change request, although this may not be universally true. Where the entry point is a ticket/change request we suggest that consistent naming of branches for ease of tracking. The change request will typically come from the dev team, based on work done in their currently non-puppet environment.

The person to whom the ticket is assigned should clone the repository to a working directory. We propose that Puppet developers have an environment configured on the test Puppet master specific to that developer, or branch you are working on. This enables developers to branch, commit changes and test their code without interfering with work that others are doing in their environments. Pushing those branches back to the repo gives others visibility of what's happening elsewhere.

```
[root@puppet ~]# cd /etc/puppetlabs/puppet/env/
[root@puppet ~]# mkdir puppet-dev
[root@puppet ~]# cd !$
[root@puppet puppet-dev]# git clone http://git@gitrepo:7990/git/PUP/puppetcode.git
Initialized empty Git repository in /etc/puppetlabs/puppet/env/puppet-dev
Password:
remote: Counting objects: 41, done.
remote: Compressing objects: 100% (33/33), done.
remote: Total 41 (delta 7), reused 0 (delta 0)
Unpacking objects: 100% (41/41), done.
```

Once the repo has been cloned, you should create a new topic branch for your changes:

```
[root@puppet puppet-dev]# git checkout -b C0000001 remotes/origin/Puppet-DEV
Branch C0000001 set up to track remote branch Puppet-DEV from origin.
Switched to a new branch 'C0000001'
```

Using topic branches is important because it logically segregates your changes from the rest of the branch you are working on. It is particularly important if your git server as it doesn't support forks of repositories to be held centrally on the server, and neither does it support pull requests from one particular copy of a repository to another. Using a branch enables you to publish your proposed changes in an isolated way to the server that enables others to download and evaluate (and merge) them. If work were done directly on the Puppet-DEV branch, it would be more complex to share proposed changes before merging, merge order would be made more difficult and tracking your work becomes more difficult.

Setting the branch to support remote tracking is useful for merging changes from the git origin whilst you are working. Assume there are multiple topic branches in flight at any point in time, and that some of those get merged into the Puppet-DEV branch before your work is complete. Given this state of affairs, it is important to monitor and merge changes happening upstream from you regularly to avoid merge conflicts. It is the responsibility of the merge manager(s) to ensure that history is not overwritten by any new merge.

The name of the branch should be meaningful – following a change ticket or trouble ticket number is good data to use for this. For other changes, an informative name is a good start. For dynamic branches, following the rules Puppet has for environment naming is important.

At this point, the branch you have created does not exist in the central version of the repository and you are working on a distributed local copy of it. All your changes should be made here.

Whilst working on your change, you can commit to your local copy as much as you like, using commit messages that are meaningful to you so you can track your progress, and if necessary revert your local branch:

```
[root@puppet modules]# git status
# On branch C0000001
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       roles/
nothing added to commit but untracked files present (use "git add" to track)
[root@puppet modules]# git add -A
[root@puppet modules]# git commit -m "adding roles module"
[C0000001 a1c8ab3] adding roles module
 7 files changed, 154 insertions(+), 0 deletions(-)
 create mode 100644 modules/roles/manifests/ds4db.pp
 create mode 100644 modules/roles/manifests/dst.pp
 create mode 100644 modules/roles/manifests/fdi.pp
 create mode 100644 modules/roles/manifests/fed.pp
 create mode 100644 modules/roles/manifests/init.pp
 create mode 100644 modules/roles/manifests/kace.pp
 create mode 100644 modules/roles/manifests/mdp.pp
```

These local commits have no impact on the central copy of the repository.

Readying the topic branch

So – I've been working on my topic branch, making changes, committing as I went along, and the time has come to push the branch to the middle for someone to review my changes and (hopefully) merge them into the Puppet-DEV branch.

At this point,the very first thing I should do is to merge any remote changes into my branch BEFORE pushing my feature branch to the repository. The remote changes will be in the Puppet-DEV branch, so I want to switch back to that branch and pull the current state of the Puppet-DEV branch into my local copy of the repo.

```
[root@puppet puppet-dev]# git checkout Puppet-DEV
Switched to branch 'Puppet-DEV'
[root@puppet puppet-dev]# git pull origin Puppet-DEV
Password:
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From http://git@gitrepo:7990/git/PUP/puppetcode
 * branch            Puppet-DEV -> FETCH_HEAD
Updating 71dbc0d..b98b8c4
Fast-forward
 duckfile |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 duckfile
```

In this way, I am brought up to date with the work other people may have done and already merged into the Puppet-DEV branch. Then, I check out my current topic branch and attempt to merge any upstream changes into it and then replay my commits on top of that new state.

```
[root@puppet puppet-dev]# git checkout C0000001
Switched to branch 'C0000001'
Your branch is ahead of 'origin/Puppet-DEV' by 1 commit.
[root@puppet puppet-dev]# git rebase Puppet-DEV
First, rewinding head to replay your work on top of it...
Applying: added moofile
[root@puppet puppet-dev]# git status
# On branch C0000001
# Your branch is ahead of 'origin/Puppet-DEV' by 2 commits.
#
nothing to commit (working directory clean)
```

This means that the hard work of conflict resolution is the responsibility of whoever merges last but leads to a better maintained repository with more meaningful messages. It also means that your work is in the correct place in the history of the repository – your changes are replayed on top of the changes in the central copy of the repository, which is what we want.

Conflict resolution

If in your topic branch you edited a file, but an upstream change also edited the file, you will probably get a merge conflict when you attempt to rebase your work. Consider the following example:

I clone and branch the repository and make changes:

```
[root@puppet tmp3]# git checkout -b conflicttest
Switched to a new branch 'conflicttest'
[root@puppet tmp]# vi conflict2
[root@puppet tmp]# git add conflict2
[root@puppet tmp]# git commit -a -m "we too have a conflict2"
[conflicttest fe8cdd4] we too have a conflict2
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 conflict2
```

Somehow, an identical change is made to the Puppet-DEV branch, which is committed and therefore we will end up with a conflict:

```
[root@puppet puppet-dev]# vi conflict2
[root@puppet puppet-dev]# git add conflict2
```

```
[root@puppet puppet-dev]# git commit -a -m "new conflict"
[Puppet-DEV 3172f9e] new conflict
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 conflict2
[root@puppet puppet-dev]# git push origin
Password:
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 278 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To http://git@gitserver:7990/git/PUP/puppetcode.git
   0bb17f8..3172f9e  Puppet-DEV -> Puppet-DEV
```

I then fetch the remote branch with a view to rebasing my work on it:

```
[root@puppet tmp]# git fetch
Password:
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From http://gitserver:7990/git/PUP/puppetcode
   0bb17f8..3172f9e  Puppet-DEV -> origin/Puppet-DEV
[root@puppet tmp]# git rebase remotes/origin/Puppet-DEV
First, rewinding head to replay your work on top of it...
Applying: Commit message for C0000008
Applying: test commit
Applying: we too have a conflict2
Using index info to reconstruct a base tree...
Falling back to patching base and 3-way merge...
Auto-merging conflict2
CONFLICT (add/add): Merge conflict in conflict2
Failed to merge in the changes.
Patch failed at 0003 we too have a conflict2

When you have resolved this problem run "git rebase --continue".
If you would prefer to skip this patch, instead run "git rebase --skip".
To restore the original branch and stop rebasing run "git rebase --abort".
```

At this point therefore I need to figure out what the appropriate content of whatever files conflict should be in order to correctly merge the upstream changes, not overwrite someone else's work and make sure that my branch has the correct code in it.

```
[root@puppet tmp]# cat conflict2
<<<<<<< HEAD
another conflict
=======
this is a different set of changes
>>>>>>> we too have a conflict2
```

Thus I need to edit the conflicting file to have the appropriate content (I may need to speak to whoevers work conflicts with my code), and change the file accordingly. Difficulty of resolving the conflict is directly proportional to how much of the file conflicts and the purpose of the change. The purpose of conflict resolution is to end up with the correct code or data.

```
[root@puppet tmp]# vi conflict2
[root@puppet tmp]# cat conflict2
this is a different set of changes
[root@puppet tmp]# git add conflict2
[root@puppet tmp]# git rebase --continue
Applying: we too have a conflict2
```

Commit squashing

Whilst I've been committing, every commit has a message:

```
[root@puppet puppet-dev]# git log
commit 7035375d3cc4f642949c0331a84f119d99b3c344
Author: git <sysadmin@organisation.com>
Date:   Mon Jun 18 15:33:37 2012 +0100
```

```
    Commit 3

commit 60208d43581db1add3487abbdbc2445dc2d639c7
Author: git <sysadmin@organisation.com>
Date:   Mon Jun 18 15:33:25 2012 +0100

    Commit 2

commit 90f223c337415368be87346e705627a32f53ded2
Author: git <sysadmin@organisation.com>
Date:   Mon Jun 18 15:33:14 2012 +0100

    Commit 1
```

In real life, I really have only one change – whatever the purpose of my topic branch was, though I made lots of commits to mark progress as I went. That information was valuable to me, but perhaps not so much to anyone else.

So – I made 3 commits – I want to *squash* them into a single commit with a meaningful message.

```
[root@puppet puppet-dev]# git rebase -i HEAD~3

pick 90f223c Commit 1
pick 60208d4 Commit 2
pick 7035375 Commit 3

# Rebase 4a15eeb..7035375 onto 4a15eeb
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

In the editor, I will want to squash 2 of the commits, so my text will look like this:

```
pick 90f223c Commit 1
squash 60208d4 Commit 2
squash 7035375 Commit 3

# Rebase 4a15eeb..7035375 onto 4a15eeb
```

Write and quit the file, and edit the combined commit message:

```
# This is a combination of 3 commits.
# The first commit's message is:
Commit 1

# This is the 2nd commit message:

Commit 2

# This is the 3rd commit message:

Commit 3

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# Not currently on any branch.
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   commit
#       new file:   mynewfile
#       new file:   mynewfile2
#
```

Would become:

```
Commit message for topic branch C0000006 for feature blah

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# Not currently on any branch.
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   commit
#       new file:   mynewfile
#       new file:   mynewfile2
#
```

At this point, my git log will look like:

```
[root@puppet puppet-dev]# git log
commit cb09cb8e23d5225e2993593c910483c2c1dc6f3f
Author: git <sysadmin@organisation.com>
Date:   Mon Jun 18 15:33:14 2012 +0100

    Commit message for topic branch C0000006 for feature blah
```

Putting the branch into the central repository makes your updates visible to other people editing code, and gives the merge master the opportunity to review changes before attempting a merge.

To push your branch to the git repository simply:

```
[root@puppet puppet-dev]# git push origin C0000008
Password:
Counting objects: 3, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 238 bytes, done.
Total 2 (delta 1), reused 0 (delta 0)
To http://git@gitserver:7990/git/PUP/puppetcode.git
 * [new branch]      C0000008 -> C0000008
```

This pushes your up to date tree to the central repository ready for the Merge master to perform the merge of your work with the Puppet-DEV branch.

Merging branches:

Merge ordering

Git repositories are incremental, and the state of the whole repository moves forward with each commit. Managing the order of merges becomes important therefore in situations where incremental changes to the repository are required (for example several different change requests), and they have to be in the correct order. This may become confusing in cases where change request number ordering does not actually indicate the order that work is being done in, or the order of changes being available to merge.

The merge master therefore needs to have a view on upcoming changes to determine:

- Order that change needs to be applied to an environment
- Order that change is ready to be merged into the repository

The order that change needs to be merged is actually determined by the order that change needs to be applied to the environment so events can happen in a deterministic and ordered manner.

Consider the following merges to the Puppet-DEV branch:

Change number	Date to be applied	Date of readiness to merge
C0000001	10/6/2012 (before C0000003)	5/6/2012
C0000002	8/6/2012	6/6/2012
C0000003	10/6/2012	7/6/2012
C0000004	10/6/2012 (ordering unimportant)	4/6/2012

In the preceding example, what we would expect to end up with is an ordering of merges like the following:

Change number	Actual merge date	Tag number
C0000001+C0000004	7/6/2012	2012060700-C0000001:C0000004
C0000002	6/6/2012	2012060600-C0000002
C0000003	7/6/2012	2012060701-C0000003

See the section below on tags for a detailed explanation of tag serial numbers.
This scenario will also have implications for topic branches – they will need to be rebased to the current state of the Puppet-DEV after each previous merge in order to preserve history.

Consider the following, where we have updates in Puppet-DEV:

```
[root@puppet puppet-dev]# git branch
  C0000001
  C0000002
  C0000003
* Puppet-DEV
  master
```

I want to merge the topic branches in the order they are shown, so I need to rebase, merge, rinse and repeat.

```
[root@puppet puppet-dev]# git log
commit 5cceac84466b64c2433759b5682b9f517ab8321c
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:12:49 2012 +0100

    Puppet-DEV - add newmodule

[root@puppet puppet-dev]# git checkout C0000001
Switched to branch 'C0000001'
Your branch and 'origin/Puppet-DEV' have diverged,
and have 1 and 1 different commit(s) each, respectively.
[root@puppet puppet-dev]# git rebase Puppet-DEV
First, rewinding head to replay your work on top of it...
Applying: C0000001 - adding module
[root@puppet puppet-dev]# git status
# On branch C0000001
# Your branch is ahead of 'origin/Puppet-DEV' by 1 commit.
#
nothing to commit (working directory clean)
[root@puppet puppet-dev]# git log
commit 454f936758d344fd537e4f2055a09694b45cda3d
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:13:52 2012 +0100

    C0000001 - adding module

commit 5cceac84466b64c2433759b5682b9f517ab8321c
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:12:49 2012 +0100

    Puppet-DEV - add newmodule
```

At this point I can attempt to merge the topic branch into Puppet-DEV

```
[root@puppet puppet-dev]# git checkout Puppet-DEV
Switched to branch 'Puppet-DEV'
[root@puppet puppet-dev]# git merge C0000001
Updating 5cceac8..454f936
Fast-forward
 modules/C0000001/manifests/init.pp |    4 ++++
 1 files changed, 4 insertions(+), 0 deletions(-)
 create mode 100644 modules/C0000001/manifests/init.pp
```

Wahoo! Let's push the change and do the next change.

```
[root@puppet puppet-dev]# git push origin
Password:
Counting objects: 8, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 456 bytes, done.
Total 6 (delta 2), reused 0 (delta 0)
To http://git@gitserver:7990/git/PUP/puppetcode.git
   5cceac8..454f936  Puppet-DEV -> Puppet-DEV
[root@puppet puppet-dev]# git checkout C0000002
Switched to branch 'C0000002'
Your branch and 'origin/Puppet-DEV' have diverged,
and have 1 and 2 different commit(s) each, respectively.
[root@puppet puppet-dev]# git rebase Puppet-DEV
First, rewinding head to replay your work on top of it...
Applying: C0000002 - added module
[root@puppet puppet-dev]# git log
commit 7ba96646d34126c5755336432ab543a598bbd6a0
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:14:46 2012 +0100

    C0000002 - added module

commit 454f936758d344fd537e4f2055a09694b45cda3d
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:13:52 2012 +0100

    C0000001 - adding module

commit 5cceac84466b64c2433759b5682b9f517ab8321c
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:12:49 2012 +0100

    Puppet-DEV - add newmodule

[root@puppet puppet-dev]# git checkout Puppet-DEV
Switched to branch 'Puppet-DEV'
[root@puppet puppet-dev]# git merge C0000002
Updating 454f936..7ba9664
Fast-forward
 modules/C0000002/manifests/init.pp |    3 +++
 1 files changed, 3 insertions(+), 0 deletions(-)
 create mode 100644 modules/C0000002/manifests/init.pp
```

Push, rinse, repeat

```
[root@puppet puppet-dev]# git checkout C0000003
Switched to branch 'C0000003'
Your branch and 'origin/Puppet-DEV' have diverged,
and have 1 and 3 different commit(s) each, respectively.
[root@puppet puppet-dev]# git log
commit c75bb6f356f26caaeacdbb70dbf84a046822b358
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:15:27 2012 +0100

    C0000003 - adding module

commit 3172f9e497917a935f7cda3daa1967324791e52c
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 14:15:20 2012 +0100
```

```
    new conflict
[root@puppet puppet-dev]# git rebase Puppet-DEV
First, rewinding head to replay your work on top of it...
Applying: C0000003 - adding module
[root@puppet puppet-dev]# git log
commit 9f28044c25c527b1e5ff33cbafb768431635b2b2
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:15:27 2012 +0100

    C0000003 - adding module

commit 7ba96646d34126c5755336432ab543a598bbd6a0
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:14:46 2012 +0100

    C0000002 - added module

commit 454f936758d344fd537e4f2055a09694b45cda3d
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:13:52 2012 +0100

    C0000001 - adding module

commit 5cceac84466b64c2433759b5682b9f517ab8321c
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:12:49 2012 +0100

    Puppet-DEV - add newmodule

[root@puppet puppet-dev]# git checkout Puppet-DEV
Switched to branch 'Puppet-DEV'
[root@puppet puppet-dev]# git merge C0000003
Updating 7ba9664..9f28044
Fast-forward
 modules/C0000003/manifests/init.pp |    3 +++
 1 files changed, 3 insertions(+), 0 deletions(-)
 create mode 100644 modules/C0000003/manifests/init.pp
[root@puppet puppet-dev]# git push origin
Password:
Counting objects: 8, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 449 bytes, done.
Total 6 (delta 2), reused 0 (delta 0)
To http://git@gitserver:7990/git/PUP/puppetcode.git
   7ba9664..9f28044  Puppet-DEV -> Puppet-DEV
```

The final git log looks like this, with proper merge ordering.

```
[root@puppet puppet-dev]# git log
commit 9f28044c25c527b1e5ff33cbafb768431635b2b2
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:15:27 2012 +0100

    C0000003 - adding module

commit 7ba96646d34126c5755336432ab543a598bbd6a0
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:14:46 2012 +0100

    C0000002 - added module

commit 454f936758d344fd537e4f2055a09694b45cda3d
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:13:52 2012 +0100

    C0000001 - adding module

commit 5cceac84466b64c2433759b5682b9f517ab8321c
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 15:12:49 2012 +0100

    Puppet-DEV - add newmodule
```

At any point in the previous example, I could have merged commits with the master branch, tagged the commit combination (for example `commit 7ba96646d34126c5755336432ab543a598bbd6a0` as a combination of modules C0000001 and C0000002) I will want to use in a Puppet environment, and then carried on.

The merge

The merge master should clone the project to a working directory and make sure he can check out the topic branch (C0000001) and the Puppet-DEV branch.

We always merge into the Puppet-DEV branch before pushing code further up the into the master branch.

In the fresh cloned repo, checkout the Puppet-DEV branch

```
[root@puppet modules]# git checkout Puppet-DEV
Switched to branch 'Puppet-DEV'
```

It is best practice to review the incoming changes before attempting to merge them. Running git diff from the current branch gives raw diff output so you can see the exact changes

```
[root@puppet modules]# git diff C0000001
diff --git a/modules/roles/manifests/ds4db.pp b/modules/roles/manifests/ds4db.pp
deleted file mode 100644
index c91c847..0000000
--- a/modules/roles/manifests/ds4db.pp
+++ /dev/null
@@ -1,30 +0,0 @@
-# == Class: roles::ds4db
.....
```

Depending on the extent of the changes, there may be significant changes to review.

Once you have reviewed the changes, it's time to merge them. Ensure you are in the correct branch:

```
[root@puppet modules]# git branch
  C0000001
* Puppet-DEV
  Master

[root@puppet modules]# git merge C0000001
Updating 7df0241..63dbbec
Fast-forward
 modules/roles/manifests/ds4db.pp |   30 ++++++++++++++++++++++++++++++
 modules/roles/manifests/dst.pp   |   30 ++++++++++++++++++++++++++++++
 modules/roles/manifests/fdi.pp   |   30 ++++++++++++++++++++++++++++++
 modules/roles/manifests/fed.pp   |    2 ++
 modules/roles/manifests/init.pp  |    6 ++++++
 modules/roles/manifests/kace.pp  |   30 ++++++++++++++++++++++++++++++
 modules/roles/manifests/mdp.pp   |   28 ++++++++++++++++++++++++++++
 7 files changed, 156 insertions(+), 0 deletions(-)
 create mode 100644 modules/roles/manifests/ds4db.pp
 create mode 100644 modules/roles/manifests/dst.pp
 create mode 100644 modules/roles/manifests/fdi.pp
 create mode 100644 modules/roles/manifests/fed.pp
 create mode 100644 modules/roles/manifests/init.pp
 create mode 100644 modules/roles/manifests/kace.pp
 create mode 100644 modules/roles/manifests/mdp.pp
```

If there are conflicts, you have the opportunity to remediate the conflicts. Once you are happy, you can commit the changes:

```
[root@puppet puppet-dev]#  git commit -a -m "merged topic branch C0000001 into Puppet-DEV"
```

Then, push the changes up to the central copy of the repository

```
[root@puppet modules]# git push origin
```

At this point you may inspect the Puppet-DEV branch in the your web console for sanity if you have one.

Deleting topic branches

Once you have successfully merged a topic branch and you will not be requiring it any more, you should delete it (otherwise you'll end up with a billion useless and confusing branches).

```
[root@puppet puppet-dev]# git branch -d C0000001
[root@puppet puppet-dev]# git push origin :C0000001
Password:
To http://git@gitserver:7990/git/PUP/puppetcode.git
 - [deleted]         C0000001
```

Moving content from Puppet-dev to the master branch.

The order of content being managed through environments might be:

Puppet-DEV => QA => UAT => Production/DR

At this stage therefore, we need to merge our Puppet-DEV changes into the master branch, and create a tag to reference the commit.

To do this, the merge master should have a clean up to date version of the repository checked out with the latest changes merged into the Puppet-DEV branch.

Check out the master branch, check that it is at the point you expect:

```
[root@puppet tmnp2]# git checkout master
Branch DEV set up to track remote branch master from origin.
Switched to a new branch 'master'
[root@puppet tmnp2]# git log
commit 7df024115d478108d2d0368b6eae55cbfb9688a7
Author: git <sysadmin@organisation.com>
Date:   Mon Jun 18 11:14:45 2012 +0100

    initial commit
```

Check out the Puppet-DEV branch, and make sure it is also what you expect:

```
[root@puppet tmnp2]# git checkout Puppet-DEV
Branch Puppet-DEV set up to track remote branch Puppet-DEV from origin.
Switched to a new branch 'Puppet-DEV'
[root@puppet tmnp2]# git log
commit e44f9834e8d74caa7ce6c483d6955f80129df899
Author: git <sysadmin@organisation.com>
Date:   Mon Jun 18 13:29:48 2012 +0100

    updates to init.pp

commit 63dbbecbbe014a1c3a383d3900d2110c29a6d781
Author: git <sysadmin@organisation.com>
Date:   Mon Jun 18 11:36:51 2012 +0100

    adding roles module

commit 7df024115d478108d2d0368b6eae55cbfb9688a7
Author: git <sysadmin@organisation.com>
Date:   Mon Jun 18 11:14:45 2012 +0100
```

```
    initial commit
```

Switch back to the master branch and run a diff, to ensure the changes you expect are the ones that you will merge in:

```
[root@puppet tmnp2]# git checkout master
Switched to branch 'master'
[root@puppet tmnp2]# git diff Puppet-master
diff --git a/modules/hash/manifests/init.pp b/modules/hash/manifests/init.pp
index df6238f..8d132e4 100644
--- a/modules/hash/manifests/init.pp
+++ b/modules/hash/manifests/init.pp
@@ -1,5 +1,5 @@
 class hash {
-# This class only provides data
+
 $applications = {
...
```

Once you are satisfied with the changes, merge the new changes in to the master branch:

```
[root@puppet tmnp2]# git merge Puppet-DEV
Updating 7df0241..e44f983
Fast-forward
 modules/hash/manifests/init.pp   |    2 +-
 modules/roles/manifests/ds4db.pp |   30 ++++++++++++++++++++++++++++++
 modules/roles/manifests/dst.pp   |   30 ++++++++++++++++++++++++++++++
 modules/roles/manifests/fdi.pp   |   30 ++++++++++++++++++++++++++++++
 modules/roles/manifests/fed.pp   |    2 ++
 modules/roles/manifests/init.pp  |    6 ++++++
 modules/roles/manifests/kace.pp  |   30 ++++++++++++++++++++++++++++++
 modules/roles/manifests/mdp.pp   |   28 ++++++++++++++++++++++++++++
 8 files changed, 157 insertions(+), 1 deletions(-)
 create mode 100644 modules/roles/manifests/ds4db.pp
 create mode 100644 modules/roles/manifests/dst.pp
 create mode 100644 modules/roles/manifests/fdi.pp
 create mode 100644 modules/roles/manifests/fed.pp
 create mode 100644 modules/roles/manifests/init.pp
 create mode 100644 modules/roles/manifests/kace.pp
 create mode 100644 modules/roles/manifests/mdp.pp
```

If the merge ran correctly, push the changes up to the server copy of the repository:

```
[root@puppet tmnp2]# git push origin
Password:
Total 0 (delta 0), reused 0 (delta 0)
To http://git@gitserver:7990/git/PUP/puppetcode.git
   7df0241..e44f983  master -> master
```

Once this is complete, then the remaining step is to update the branch with a tag to reference that particular commit that you can use to check out the right version of code for a particular environment

Production tags

When code is merged into the master branch, tag the commit with a useful name or serial number.

From the point of view of usability and ordering (you see tags in serial number order), BIND like serial numbers are good for this purpose. Using something else to index a tag is fine, but be wary of situations where tags might not be incremental despite ordering.

YYYYMMDDCC-<CHANGE NUMBER>(:<CHANGE NUMBER>)

Where:

```
        Y                       = year
        M                       = month,
        D                       = day,
        C                       = change number (starting at a 0 index)
        <CHANGE NUMBER>         = change/fault numbers delimited by a :
```

Thus, you can have human accessible references to a particular production commit that you can refer to.

```
[root@puppet puppet-dev]# git tag -a 2012061905 -m "changes to production code"
[root@puppet puppet-dev]# git push origin --tags
Password:
Counting objects: 1, done.
Writing objects: 100% (1/1), 170 bytes, done.
Total 1 (delta 0), reused 0 (delta 0)
To http://git@gitserver:7990/git/PUP/puppetcode.git
 * [new tag]         2012061905 -> 2012061905
```

To list tags, simply type

```
[root@puppet prod]# git tag
2012061900
2012061901
2012061902
2012061903
2012061904
2012061905
```

To find out about a specific tag

```
[root@puppet prod]# git show 2012061905
tag 2012061905
Tagger: git <sysadmin@organisation.com>
Date:   Tue Jun 19 10:35:47 2012 +0100

changes to production code

commit f98140d3e5cad6edf1af263bb891553be5fee2ef
```

To tag an earlier commit,

```
[root@puppet tmp]# git log
commit 3e01b2196c53880c5c8e161214e08627b2183206
Merge: 7f068cf fb3af9e
Author: git <sysadmin@organisation.com>
Date:   Tue Jun 19 13:58:33 2012 +0100

    Merge branch 'Puppet-DEV' of http://gitserver:7990/git/PUP/puppetcode into Puppet-DEV

[root@puppet tmp]# git tag -a 2012061920-C0000002 3e01b2196c53880c5c8e161214e08627b2183206
[root@puppet tmp]# git tag
2012061920-C0000002
```

Using a change request number to tag a commit seems fine, but there may be dragons in the event that merge order into the repository does not follow change numbers.

C000005 is merged after C000010, but C000010 looks like the later tag. I check out C000010, and I lose the changes I needed for change C00005 because C000010 is actually an earlier commit. Badness ensues.

Checking out code to an environment

New environment

If the repository is not yet checked out into the appropriate filesystem location, check it out:

```
[root@puppet puppet-dev]# git clone http://git@gitserver:7990/git/PUP/puppetcode.git .
Initialized empty Git repository in /etc/puppetlabs/puppet/env/puppet-dev/.git/
Password:
remote: Counting objects: 51, done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 51 (delta 10), reused 0 (delta 0)
Unpacking objects: 100% (51/51), done.
[root@puppet puppet-dev]# git branch
* master
```

Then switch to the appropriate tag or branch (Puppet-DEV or a branch)

Puppet-dev

The puppet-dev environment should always be kept up to date with the latest commit merged in by the merge master to the Puppet-DEV branch of the git repo.

Ensure you are on the Puppet-DEV branch, and pull from the Stash repository:

```
[root@puppet tmp]# git branch
* Puppet-DEV
[root@puppet tmp]# git pull origin Puppet-DEV
Password:
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.
From http://gitserver:7990/git/PUP/puppetcode
 * branch            Puppet-DEV -> FETCH_HEAD
Merge made by recursive.
 cheese     |    1 +
 newfile    |    1 +
 2 files changed, 2 insertions(+), 0 deletions(-)
 create mode 100644 Puppet-DEV
 create mode 100644 cheese
 create mode 100644 newfile
```

This will put the environment to the latest commit in Puppet-DEV. If you want to move back in time, check out an earlier, alternative commit. If there isn't a tag, you can use the full commit reference (e.g. `e9e3abefb9352ea0b327451942ebdd88ca139437`)

UAT, QA, PRODUCTION/DR

These environments might be pinned to specific commits made to the master branch, as referenced by tags in that branch of the git repository, or they may just track HEAD

```
[root@puppet prod]# git fetch origin
Password:
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From http://gitserver:7990/git/PUP/puppetcode
   f98140d..46ed5b1  master     -> origin/master
 * [new tag]         2012061906 -> 2012061906
[root@puppet prod]# git status
# On branch Puppet-DEV
# Your branch is behind 'origin/Puppet-DEV' by 2 commits, and can be fast-forwarded.
#
nothing to commit (working directory clean)
```

At this point you should simply checkout the latest tag:

```
[root@puppet prod]# git checkout 2012061906
Previous HEAD position was f98140d... added goatfile
HEAD is now at 46ed5b1... added a real horse
```

Checking the current state of an environment

Git describe can be used to determine the current tag checked out to an environment:

```
[root@puppet prod]# git tag
2012061900
2012061901
2012061902
2012061903
2012061904
2012061905
2012061906
2012061907
2012061908
[root@puppet prod]# git describe --exact-match
2012061908
[root@puppet prod]# git checkout 2012061903
Previous HEAD position was d053be1... added some content
HEAD is now at deba543... added moofile
[root@puppet prod]# git describe --exact-match
2012061903
```

If your current HEAD doesn't correspond to a tagged commit, you will get an error message:

```
[root@puppet puppet-dev]# git describe --exact-match
fatal: no tag exactly matches 'c08f4f5a251c03988757d0fa257143dd4a87a3af'
```

Similarly, if your current local state has been modified from the checkout, then you will see an error message:

```
[root@puppet puppet-dev]# git status
# Not currently on any branch.
# Untracked files:
#    (use "git add <file>..." to include in what will be committed)
#
#        moo
[root@puppet puppet-dev]# git commit -a -m "moo"
[detached HEAD f23cfc7] moo
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 moo
[root@puppet puppet-dev]# git describe --exact-match
fatal: no tag exactly matches 'f23cfc762dccbd5fe7b70f8ba766d8c177b6bda0'
```

Reverting changes.

Changes to the central repository should always be cumulative. Backing out a change is NOT and should NEVER be cases of running git revert on something you have already pushed to the central repository. Note – you may run a git revert on commits you have done in a topic branch before merging into Puppet-DEV, but once merged you must never revert commits. This is known as rewriting history and is bad because other people are probably at that point relying on your history.

Undoing a change to production machines in the event that bad code made it out into the wild should be as simple as re-checking out the previous known good commit (as referenced by a tag) – this will wind the repository for that environment back to a (presumed) good pre-change code to give you the opportunity to fix whatever error you encountered before re-committing code and updating the Puppet environment.

Once you have remediated any errors with the code (i.e. new topic branch, merge into Puppet-DEV, merge into master, new tag) you can simply re-fetch the data from origin and it will bring you back up to date:

```
[root@puppet prod]# git checkout 2012061906
HEAD is now at 46ed5b1... added a real horse
```

Meanwhile, chaos occurs – you decide to roll back the change:

```
Previous HEAD position was 46ed5b1... added a real horse
HEAD is now at f98140d... added goatfile
```

So, the code change is examined, some new development happens, there are some commits and eventually there's a new tag to check out.

```
[root@puppet prod]# git fetch
Password:
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From http://gitserver:7990/git/PUP/puppetcode
   46ed5b1..e92d4f2  master     -> origin/master
 * [new tag]         2012061907 -> 2012061907
[root@puppet prod]# git checkout 2012061907
Previous HEAD position was f98140d... added goatfile
HEAD is now at e92d4f2... chickens
```

Changing live code

Because checking out tags puts you in a situation where you aren't in a branch, in order to make changes you need to convert your directory to a branch. You are much better off not doing this and making the changes under controlled conditions elsewhere, fetching and checking out specific code revisions. If you do need to:

```
[root@puppet prod]# git checkout -b C0000002
Switched to a new branch 'C0000002'
[root@puppet prod]# git status
# On branch C0000002
nothing to commit (working directory clean)
[root@puppet prod]# vi chuckfile
[root@puppet prod]# git status
# On branch C0000002
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   chuckfile
#
no changes added to commit (use "git add" and/or "git commit -a")
[root@puppet prod]# git commit -a -m "changed the chickens"
[C0000002 8ef8bb2] changed the chickens
 1 files changed, 1 insertions(+), 1 deletions(-)
[root@puppet prod]# git push origin C0000002
Password:
Counting objects: 5, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 278 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To http://git@gitserver:7990/git/PUP/puppetcode.git
 * [new branch]      C0000002 -> C0000002
```

Then push and merge the changes as normal, fetch the repo back and recheck out the checked commits. If you are running an automated task to check out git, check branches and commits, making this kind of live change will probably require tinkering with that process.