

   RSS for: [front page](#) | [all entries](#) | [comments](#)

Search...



[Structured Procrastination](#)

because there's always something more interesting than what you should be doing

- [Home](#)
- [music](#)
- [geek](#)
- [hacks](#)
- [training](#)
- [work](#)
- [fun](#)
- [travel](#)

[Easier upstreaming / back-porting of patch series with git](#)

By [Adam](#), September 19, 2013 9:22 pm

Have you ever needed to port a selection of commits from one [git](#) branch to another, but without doing a full merge? This is a common challenge, e.g.

- forward-porting / upstreaming bugfixes from a stable release branch to a development branch, or
- back-porting features from a development branch to a stable release branch.

Of course, git already goes quite some way to making this possible:

- [git cherry-pick](#) can port individual commits, or even a range of commits (since git 1.7.2) from anywhere, into the current branch.
- [git cherry](#) can compare a branch with its upstream branch and find which commits have been upstreamed and which haven't. This command is particularly clever because, thanks to [git patch-id](#), it can correctly spot when a commit has been upstreamed, even when the upstreaming process resulted in changes to the commit message, line numbers, or whitespace.
- [git rebase --onto](#) can transplant a contiguous series of commits onto another branch.

It's not always that easy ...

However, on the occasions when you need to sift through a larger number of commits on one branch, and port them to another branch, complications can arise:

- If cherry-picking a commit results in changes to its patch context, `git patch-id` will return a different SHA-1, and subsequent invocations of `git cherry` will incorrectly tell you that you haven't yet ported that commit.
- If you mess something up in the middle of a `git rebase`, recovery can be awkward, and `git rebase --abort` will land you back at square one, undoing a lot of your hard work.
- If the porting process is big enough, it could take days or even weeks, so you need some way of reliably tracking which commits have already been ported and which still need porting. In this case you may well want to adopt a divide-and-conquer approach by sharing out the porting workload between team-mates.
- The more the two branches have diverged, the more likely it is that conflicts will be encountered during cherry-picking.
- There may be commits within the range you are looking at which after reviewing, you decide should be excluded from the port, or at least porting them needs to be postponed to a later point.

It could be argued that all of these problems can be avoided with the right branch and release management workflows, and I don't want to debate that in this post. However, this is the real world, and sometimes it just happens that you have to deal with a porting task which is less than trivial. Well, that happened to me and my team not so long ago, so I'm here to tell you that I have written and published some tools to solve these problems. If that's of interest, then read on!

Building tools around `git notes`

Since `git patch-id` doesn't guarantee to return an SHA-1 which is stable enough for us to always depend on, we need some other way of reliably tracking which commits have been upstreamed, and where upstreamed commits came from. It would also be useful to be able to mark selected commits as *blacklisted*, i.e. permanently excluded

from the current porting process, or at least as *deferred*, i.e. *temporarily* excluded.

Of course, we could do that simply by maintaining an out-of-band text file somewhere outside the repositories in question, but that doesn't integrate at all with existing git workflows. Whilst pondering these challenges, I discovered a little-known and seldom-used feature of git, which is the ability to attach arbitrary text as notes on a given commit, via the [git notes](#) command. "This sounds perfect!", I thought, and actually as it turned out, it served as a pretty nice foundation for some higher-level tools which needed building:

- [git-icing](#) – an "interface-candy" wrapper around git cherry which adds a splash of colour, and checks for specially formatted notes attached to commits which indicate that those commits are temporarily or permanently excluded from the upstreaming backlog
- [git-cherry-menu](#) – an interactive wrapper around `git-icing` and `git-notes` which makes it easy to cherry-pick and/or blacklist non-upstreamed commits
- [git-rnotes](#) – a wrapper around `git notes` which makes it easier to share notes to and from remote repositories

A "real world" example

Let's see how this works in practice, by pretending we need to backport a bunch of commits from [OpenStack Nova](#)'s `master` branch to its `stable/grizzly` release branch.

```
$ git clone git@github.com:openstack/nova.git
Cloning into 'nova'...
remote: Counting objects: 208984, done.
remote: Compressing objects: 100% (54766/54766), done.
remote: Total 208984 (delta 167220), reused 188334 (delta 148279)
Receiving objects: 100% (208984/208984), 124.50 MiB | 4.42 MiB/s, done.
Resolving deltas: 100% (167220/167220), done.
$ cd nova
```

Now, `master` has changed a **lot** since `grizzly` was released, so let's keep our pretend project to a manageable size by only attempting to backport only commits which appeared in the first few days after the stable branch was established:

```
$ git describe $( git merge-base origin/stable/grizzly master )
2013.1.rc1
$ git show 2013.1.rc1 | head -n3
tag 2013.1.rc1
Tagger: Thierry Carrez <thierry@CENSORED.org>
Date:   Wed Mar 20 17:13:32 2013 +0100
```

OK, so `grizzly` was branched at the point where the first release candidate was tagged. That makes sense. Now let's choose a point about 5 days later:

```
$ git rev-list --no-merges --before='25 March 2013' origin/stable/grizzly..mas
d6a9d38a0b7488dd77d3a01ae9c4171c271d7314
$ git tag to-backport d6a9d38
```

... and get ready to backport commits from `to-backport` back into our local `stable/grizzly` branch:

```
$ git checkout stable/grizzly
Branch stable/grizzly set up to track remote branch stable/grizzly from origin
Switched to a new branch 'stable/grizzly'
```

(If you are following along at home, your `rev-list` will give a different result, since `origin/stable/grizzly` will have changed since I wrote this. Don't worry – just copy the `git tag` command above verbatim, and create your local `grizzly` branch via `git checkout -b stable/grizzly fc9af8f0a9` so that it points to the same place mind did when writing this article.)

At this point, two quick clarifications are needed, because things can get a bit confusing.

A quick aside: some observations on terminology

Firstly, if you read the [git cherry man page](#) or `-h` usage text, it is described as a tool to “find commits not merged upstream”, with the expectation that you would cherry-pick the commits it finds *onto* the upstream branch. But if you happen to be backporting, then the cherry-picking will happen in the *opposite* direction, so the “upstream” referred to by the man page is actually the real downstream, and vice-versa. Confused yet? If so, wherever you read the word “upstream” in the `git cherry`

documentation / help text, pretend that you actually read “target”. Then just think of “source” and “target” branches, and remember that the process is always to look for commits in the source which are not yet in the target, and then cherry-pick those into the target (or decide that you don’t want to). The same applies for the `git icing` and `cherry-menu` help text.

Secondly, normally we’d be porting from one branch to another, not from a tag to a branch. But in this artificially constructed scenario, it wouldn’t make much sense to create a `to-backport` *branch* pointing to `d6a9d38`, because that branch isn’t going to move; the only branch which will be changing is our `stable/grizzly` branch, as we cherry-pick stuff into it. In other words, the target has to be a branch, because that will change during the cherry-picking, but the source can be a branch / tag / ref / commit SHA-1 ... basically anything which [git rev-parse](#) understands.

How far behind?

When embarking on a porting project, the first question is usually “how much work is this going to be?” Fortunately `git-icing` is here to help!

(By the way, if you want to follow along with the below by trying this yourself, simply download the three scripts either from the links above or

```
git clone https://github.com/aspiers/git-config
```

and then copy them to any directory on your `$PATH`, making sure that they are executable.)

`git-icing` is designed to be invoked in exactly the same way as `git cherry`, but it has some extra options, including the convenient `--summary` option, and a verbosity parameter which controls which categories of commits are displayed:

```
$ git icing -h
```

```
usage: git icing [options] [git cherry args]
```

```
-v, --verbosity [N]
```

```
Set verbosity level
```

```
-s, --summary
```

```
Show summary
```

first field	verbosity level	description of classification
+	1	not yet upstream
!	1	still needs to be upstreamed, tracked on a TODO list
.?	1	upstream *and* blacklisted?!
!?	1	upstream *and* marked as a TODO?
.	2	blacklisted - should not be pushed to this upstream
?	2	not yet upstream, with unparseable note
-	3	already upstream
#	3	already upstream with an annotation note

(When you run that on a colour-capable terminal, you'll also see that the classifications are also nicely colour-coded so that commits which require attention naturally capture your eye more than those which don't.)

Let's see all commits (basically a colour-coded version of what `git cherry` would show us), plus a summary of the status quo:

```
$ git icing -v3 --summary stable/grizzly to-backport
+ 348cfedfcadc377fa91029b932b2a93d16475822 nova-manage: remove redundant 'dest
+ f16534db22d63ce9f762ee6f7a9245126e9f28ed nova-manage: remove unused import
- 775e3b02b2afbfb101db22b87a1c3b189d68532e1 Pass project id in quantum driver s
+ dbbbbfbf4ecb2c7b03ebb30fa11e077d35b2186cbb Enable tox use of site-packages for
- 76844433f69a7c29ed4566ad34d7e9740feaf660 Add caching for ec2 mapping ids.
+ def5fa1e0e1c0426d973e4d8e3935c6eb18698dc Set version to 2013.2
+ 3aa80834e6aab2456f0f5229a63c2dcef007cb36 Change type of ssh_port option from
+ 26aa01094a79939320d58f2fe2d5731f169987b1 Change arguments to volume_detach()
+ b52a2157bedba693c5da7dcb783b7a151769d6b1 Add placeholder migrations to allow
+ 8d5159dd6dd36d6f3d1aadcd1123574c5b0aa0614 Don't actually connect to libvirt
+ 8cd5f862ac071023d5a7984f07513e5aa91f7d3d Bring back sexy colorized test resu
- 45e65d8c0301da689de1afcbc9f45756e71097ab Update instance network info cache
- b8f9815c5ab61466009c0447f54abc4c309e4e3f nova-manage vm list fails looking '
+ c199fd6fa39311305b5b5d94c2e3732b826a6414 Remove outdated try except block in
- 03a2463be8034ee4764ac97e8020fc3d3a32f1fd Make nova.virt.fake.FakeDriver usea
- 67628c56caf9d84588a92448880ecdb33eea08b4 Fixes passing arbitrary conductor_a
+ e136de1aea8d1469465272942fa3d2769cbe3a80 xenapi: fix console for rescued ins
+ 5df18f6b915cf49ac8559274a631f1366eab00fd Add a comment to placeholder migrat
- 9075069098e32b47bd5011e2653a23b61c18d4a3 Initialize compute manager before l
- 3801a4d2f4c59dbfda49131ddde22fcb3976d651 translate cinder BadRequest excepti
- 132a0c1fd1eb127e393e5794ffa6d4a3a4950567 Don't log traceback on rpc timeout.
```

```
+ 81204d4020712cf5b5b368b75d934d637b2c001b Sync rpc from oslo-incubator.
+ f665d798234c19cccc148a178e94c3717ba8bd6e Sync everything from oslo-incubator
- 12f7d6332a731421d6e2a190ac60828c1cf98910 Reset ec2 image cache between S3 te
+ 7a9ce26aaf03459a7ea367fbdb91d91626a1ee93 py2.6 doesn't support TextTestRunne
- 5908b60b0420f1ad528e56b0c147a330e9a1a5d6 Make _downsize_quota_delta() use st.
+ 91b9c208f0c6e21759c7828aa42b3216853be425 disable colorizer as it swallows fa
+ df0560f0353bc0837cc62cfaa9029d21d45c529e Imported Translations from Transife
+ d6a9d38a0b7488dd77d3a01ae9c4171c271d7314 set up FakeLogger for root logger.
```

Summary

=====

```
11 commits processed:
    11 already upstream
```

```
18 commits remaining:
    18 not yet upstream
```

Progress: 11 / 29 commits (37%)

OK, not bad! We didn't even do any real work yet, we're already 37% of the way through. That's because some of the commits (the ones prefixed by -) have already been cherry-picked into `stable/grizzly`. Let's prove this for the first commit prefixed by - in the above output, which was `775e3b02`:

```
$ git log -n1 775e3b02
commit 775e3b02b2afb101db22b87a1c3b189d68532e1
Author: Kieran Spear <kispear@CENSORED.com>
Date:   Mon Mar 18 17:32:26 2013 +1100
```

Pass project id in quantum driver secgroup list

The quantum driver is always returning security groups from every tenant a user has access to, even when the "project" filter is supplied.

Make sure we pass along the project value when we call `quantum.list_security_groups()` so it's properly filtered.

Fixes bug 1155381.

Change-Id: I682c66a1f3f9db18b5f9924a37b45c759ff259f7

This commit appears to be only on `master`, not on `stable/grizzly`:

```
$ git branch --all --contains 775e3b02
master
remotes/origin/HEAD -> origin/master
remotes/origin/master
```

Let's try to find the commit which cherry-picked it into `stable/grizzly`:

```
$ git log --format=fuller --no-merges --grep='Pass project id' stable/grizzly
commit 989435822e5841bc1355e75bbdb003b10a3baf58
Author:      Kieran Spear <kispear@CENSORED.com>
AuthorDate:  Mon Mar 18 17:32:26 2013 +1100
Commit:      Chuck Short <chuck.short@CENSORED.com>
CommitDate:  Tue Mar 26 10:09:40 2013 -0500
```

Pass project id in quantum driver secgroup list

The quantum driver is always returning security groups from every tenant a user has access to, even when the "project" filter is supplied.

Make sure we pass along the project value when we call `quantum.list_security_groups()` so it's properly filtered.

Fixes bug 1155381.

Change-Id: I682c66a1f3f9db18b5f9924a37b45c759ff259f7

(cherry picked from commit 775e3b02b2afbf101db22b87a1c3b189d68532e1)

```
$ git branch --all --contains 9894358
* stable/grizzly
remotes/origin/stable/grizzly
```

There it is! Notice that it was committed 6 days after RC1 was tagged, but it has the same `patch-id` as the original, even though the commit message/date and committer are all different:

```
$ git patch-id < <(git show 9894358)
42dd410cf36ad1c16ceb1bb7f6ed6e3f01cc28bf 989435822e5841bc1355e75bbdb003b10a3ba
$ git patch-id < <(git show 775e3b02)
42dd410cf36ad1c16ceb1bb7f6ed6e3f01cc28bf 775e3b02b2afbf101db22b87a1c3b189d6853
```


git cherry-menu – let the picking commence

Now we can actually start porting the remaining commits. [git-cherry-menu](#) provides an interactive interface to make this process less tedious:

```
$ git cherry-menu -h
usage: git [options] cherry-menu <command> [<args>...]
suggested options:
```

```
[... snipped ...]
```

<command> <args> are typically "git icing -v3" or "git cherry", but can be anything which gives output in the same format. This allows more control over which commits constitute the upstreaming backlog, e.g.

```
git icing -v2 $upstream $downstream | grep ... > tmpfile
# Could edit tmpfile here if we want
git-cherry-menu cat tmpfile
```

Provides an interactive wrapper around git-icing (or git cherry). For each commit provided on STDIN by COMMAND which has not yet been upstreamed, asks the user whether they want to cherry-pick the commit, blacklist it, or skip it. After a successful cherry-pick, the source commit will be automatically blacklisted if the patch-id changed.

You can quit the process at any time and safely re-run it later - it will resume from where you left off.

Invoking icing with "-v2" ensures that previously blacklisted / upstreamed commits are also processed.

Let's give it a go ...

```
$ git cherry-menu git icing -v3 stable/grizzly to-backport
commit 348cfedfcadc377fa91029b932b2a93d16475822
Author: Zhiteng Huang <zhiteng.huang@CENSORED.com>
Date: Tue Feb 26 09:23:13 2013 +0800
```

```
nova-manage: remove redundant 'dest' args
```

```
Includes a hack to calculate 'dest' from the argument name. This hack is
removed in a later commit.
```

```
Change-Id: I60567ff232ab7699f3234b3bfc1618a17a648976
```

```
diff --git a/bin/nova-manage b/bin/nova-manage
index 0fde8ba..4919d88 100755
--- a/bin/nova-manage
+++ b/bin/nova-manage
@@ -127,10 +127,10 @@ def param2id(object_id):
     class VpnCommands(object):
         """Class for managing VPNs."""

-    @args('--project', dest="project_id", metavar='<Project name>',
+    @args('--project', dest='project_id', metavar='<Project name>',

[... snipped ...]

Cherry-pick / blacklist / skip 348cfedfca, or quit ?
```

It's showing us the first commit in the porting backlog, and asking us to decide whether to cherry-pick it, blacklist it (i.e. permanently exclude it from the porting process), or defer the decision by skipping it and moving onto the next commit in backlog. (q quits the whole process, of course.)

Let's try cherry-picking it:

```
Cherry-pick / blacklist / skip 348cfedfca, or quit ? c

error: could not apply 348cfed... nova-manage: remove redundant 'dest' args
hint: after resolving the conflicts, mark the corrected paths
hint: with 'git add <paths>' or 'git rm <paths>'
hint: and commit the result with 'git commit'

Spawning a shell so you can fix; exit the shell when done.
$
```

Oh dear, this one's going to need some further thought. Let's skip it for now:

```
$ git reset --hard
HEAD is now at ddb676d Merge "Typo: certicates=>certificates in nova.conf"
```

```
$ exit
```

```
Warning: HEAD did not change; no action taken.
```

```
Press enter to continue ...
```

```
-----  
commit f16534db22d63ce9f762ee6f7a9245126e9f28ed
```

```
Author: Zhiteng Huang <zhiteng.huang@CENSORED.com>
```

```
Date: Tue Feb 26 09:23:13 2013 +0800
```

```
nova-manage: remove unused import
```

```
Unused since commit 9ff3121b
```

```
Change-Id: I76bb49669d1cdfc3bf5b1c20087b4bd77420cd91
```

```
diff --git a/bin/nova-manage b/bin/nova-manage
```

```
index 4919d88..e93c3d8 100755
```

```
--- a/bin/nova-manage
```

```
+++ b/bin/nova-manage
```

```
@@ -88,7 +88,6 @@ from nova.openstack.common import timeutils
```

```
from nova import quota
```

```
from nova.scheduler import rpcapi as scheduler_rpcapi
```

```
from nova import servicegroup
```

```
-from nova import utils
```

```
from nova import version
```

```
CONF = cfg.CONF
```

```
Cherry-pick / blacklist / skip f16534db22, or quit ?
```

cherry-menu presents the next commit in the backlog, which in this case is removing an unused import from `bin/nova-manage`. Sounds pretty straight-forward, so let's cherry-pick:

```
Cherry-pick / blacklist / skip f16534db22, or quit ? c
```

```
[stable/grizzly 3f013e4] nova-manage: remove unused import
```

```
Author: Zhiteng Huang <zhiteng.huang@CENSORED.com>
```

```
1 file changed, 1 deletion(-)
```

Looks like this worked fine. The output continues:

Already upstream: 775e3b02b2 - Pass project id in quantum driver secgroup list

Here `cherry-menu` is telling us it automatically skipped over a commit which is already in our target branch. The output continues with the next commit in the backlog, which we can also successfully pick:

```
commit dbbbbf4ecb2c7b03ebb30fa11e077d35b2186cbb
Author: Clark Boylan <clark.boylan@CENSORED.com>
Date: Thu Feb 7 21:39:31 2013 -0800
```

```
    Enable tox use of site-packages for libvirt.
```

```
    Enable the use of site-packages in tox which will allow the use of the
    system install of libvirt while testing.
```

```
    Hardcode the libvirt host UUID for tests that check this UUID when
    system libvirt is being used. Without this hardcoding eight tests
    would fail when using the system libvirt install.
```

```
    Partially fixes bug #1113181
```

```
Change-Id: I59c5fbd45639962c0963298203c39759b6ca2d11
```

```
[... snipped ...]
```

```
Cherry-pick / blacklist / skip dbbbbf4ecb, or quit ? c
```

```
[stable/grizzly b4fdf14] Enable tox use of site-packages for libvirt.
Author: Clark Boylan <clark.boylan@CENSORED.com>
2 files changed, 8 insertions(+)
```

Already upstream: 76844433f6 - Add caching for ec2 mapping ids.
commit def5fae0e1c0426d973e4d8e3935c6eb18698dc
Author: Thierry Carrez <thierry@CENSORED.org>
Date: Wed Mar 20 16:21:48 2013 +0100

```
Set version to 2013.2
```

```
Open Havana development by setting version to 2013.2.
```

```
Change-Id: I37917d28a1f9e0adc2fe3e382412ebc4ed0f3bee
```

```
diff --git a/setup.py b/setup.py
index fd968ee..e4bc7d4 100644
--- a/setup.py
+++ b/setup.py
@@ -25,7 +25,7 @@
```

```
    setuptools.setup(
        name=project,
-       version=common_setup.get_version(project, '2013.1'),
+       version=common_setup.get_version(project, '2013.2'),
        description='cloud computing fabric controller',
        author='OpenStack',
        author_email='nova@CENSORED.launchpad.net',
```

```
Cherry-pick / blacklist / skip def5fale0e, or quit ?
```

Now, obviously we don't want to backport this version change, so let's blacklist it in order to permanently excluded it from the porting backlog:

```
Cherry-pick / blacklist / skip def5fale0e, or quit ? b
```

At this point, your favourite `$EDITOR` will be launched via an invocation of `git notes edit`, and you'll be presented with a text buffer which looks like this:

```
skip: all
XXX (you can optionally change the "all" above to the name of the
XXX upstream branch if you want to limit blacklisting to that upstream)

XXX Enter your justification for blacklisting here or
XXX remove the whole note to cancel blacklisting.

#
# Write/edit the notes for the following object:
```

```
# commit def5fa1e0e1c0426d973e4d8e3935c6eb18698dc
# Author: Thierry Carrez <thierry@CENSORED.org>
# Date:   Wed Mar 20 16:21:48 2013 +0100
#
#       Set version to 2013.2
#
#       Open Havana development by setting version to 2013.2.
#
#       Change-Id: I37917d28a1f9e0adc2fe3e382412ebc4ed0f3bee
#
#  setup.py | 2 +-
#  1 file changed, 1 insertion(+), 1 deletion(-)
```

At this point you should remove all the lines beginning `xxx`, and decide whether you want this commit to be blacklisted from **any** future porting operation. If so, you can leave the `skip: all` line as is and simply save the buffer and quit the editor, although it's best practice to also give a sentence or two justifying why you're blacklisting the commit.

Alternatively you can change `all` to the name of the branch to which this commit should never be ported, which in this case is `stable/grizzly`. It is also permitted to use Ruby regular expressions here, by surrounding the regexp with forward slashes, e.g.

```
skip: /stable/
```

It never makes sense to port version number changes back to stable releases.

Once the editor has been quit, we see:

```
Blacklisted def5fa1e0e
Press enter to continue ...
```

Behind the scenes, `cherry-menu` has attached a note to this commit within the `refs/notes/upstreaming` namespace:

```
$ git notes --ref=upstreaming show def5fa
skip: all
```

and you can even see the complete history of your note-editing:

```
$ git log notes/upstreaming
commit ed6cf5fa75921f23088656b7a8d8a96faa9d769d
Author: Adam Spiers <aspiers@CENSORED.com>
Date:   Fri Sep 20 15:59:04 2013 +0100
```

Notes added by 'git notes edit'

```
commit c1f6ba13e643de4da354375b479470599f92b272
Author: Adam Spiers <aspiers@CENSORED.com>
Date:   Fri Sep 20 15:44:15 2013 +0100
```

Notes added by 'git notes add'

[... snipped ...]

After all this hard work, maybe it's time for lunch. We can safely quit `cherry-menu` via `Control-C` or `q`, and resume it later. Before tucking into a tasty sandwich, let's quickly review our progress:

```
$ git icing -v3 -s stable/grizzly to-backport
+ 348cfedfcadc377fa91029b932b2a93d16475822 nova-manage: remove redundant 'dest
- f16534db22d63ce9f762ee6f7a9245126e9f28ed nova-manage: remove unused import
- 775e3b02b2afbf101db22b87a1c3b189d68532e1 Pass project id in quantum driver s
- dbbbbf4ecb2c7b03ebb30fa11e077d35b2186cbb Enable tox use of site-packages for
- 76844433f69a7c29ed4566ad34d7e9740feaf660 Add caching for ec2 mapping ids.
. def5fa1e0e1c0426d973e4d8e3935c6eb18698dc Set version to 2013.2
+ 3aa80834e6aab2456f0f5229a63c2dcef007cb36 Change type of ssh_port option from
+ 26aa01094a79939320d58f2fe2d5731f169987b1 Change arguments to volume_detach()
```

[... snipped ...]

Summary

=====

```
14 commits processed:
    13 already upstream
    1 blacklisted - should not be pushed to this upstream
```

```
15 commits remaining:
```

```
15 not yet upstream
```

```
Progress: 14 / 29 commits (48%)
```

Wow, almost half-way! Notice also how the prefix by `def5fa1` has changed to a period sign to indicate that it's now blacklisted.

After lunch, we can resume the process:

```
$ git cherry-menu git icing -v3 stable/grizzly to-backport
commit 348cfedfcadc377fa91029b932b2a93d16475822
Author: Zhiteng Huang <zhiteng.huang@CENSORED.com>
Date: Tue Feb 26 09:23:13 2013 +0800
```

```
nova-manage: remove redundant 'dest' args
```

```
[... snipped ...]
```

```
Cherry-pick / blacklist / skip 348cfedfca, or quit ?
```

This is the same commit which caused conflicts last time we tried to pick it. Let's officially postpone resolving conflicts by adding it to a "TODO" blacklist. Press `b`, then as before, remove the `xxx`, and also replace the `skip: all` line with:

```
TODO: fix conflicts next week
```

```
Blacklisted 348cfedfca
Press enter to continue ...
```

```
-----

Already upstream: f16534db22 - nova-manage: remove unused import
Already upstream: 775e3b02b2 - Pass project id in quantum driver secgroup list
Already upstream: dbbbb4ecb - Enable tox use of site-packages for libvirt.
Already upstream: 76844433f6 - Add caching for ec2 mapping ids.
Blacklisted: def5fale0e - Set version to 2013.2
commit 3aa80834e6aab2456f0f5229a63c2dcef007cb36
Author: Devananda van der Veen <devananda.vdv@CENSORED.com>
Date: Wed Mar 20 09:19:44 2013 -0700
```

```
Change type of ssh_port option from Str to Int
```



```
The type of CONF option virtual_power_ssh_port was incorrectly defaulted
to Str. This can cause Paramiko to raise when casting to %d.
```

```
Fixes bug 1157824.
```

```
Change-Id: I30ddd1ff0da45f8392085249f1bd2a539b201a7e
```

Now we are taken immediately to where we left off just before lunch, because `git-icing` automatically detected the cherry-picks we did earlier, and also that we blacklisted the 2013.2 version change commit.

If we run `git-icing` again, we'll see that it understands the significance of the `TODO` attached to the first commit in the backlog:

```
$ git icing -v3 -s stable/grizzly to-backport
! 348cfedfcadc377fa91029b932b2a93d16475822 nova-manage: remove redundant 'dest
- f16534db22d63ce9f762ee6f7a9245126e9f28ed nova-manage: remove unused import
- 775e3b02b2afbf101db22b87a1c3b189d68532e1 Pass project id in quantum driver s
```

```
[... snipped ...]
```

```
Summary
```

```
=====
```

```
14 commits processed:
```

```
    13 already upstream
```

```
    1 blacklisted - should not be pushed to this upstream
```

```
15 commits remaining:
```

```
    14 not yet upstream
```

```
    1 still needs to be upstreamed, tracked on a TODO list
```

```
Progress: 14 / 29 commits (48%)
```

On subsequent invocations of `git cherry-menu`, it will offer the option of editing existing notes. There is also an option to automatically skip any notes including the word `TODO`:

```
$ git cherry-menu -h
```

```
usage: git [<options>] cherry-menu <command> [<args>...]
```

```
suggested options:
```

```
[... snipped ...]
```

```
-c cherry-menu.skip-todos=true
```

```
    Skip commits which have notes including 'TODO'. This allows
    unresolved upstreaming tasks to be tracked via an external
    issue tracker without getting in the way during repeated
    runs of cherry-menu.
```

```
[... snipped ...]
```

So that's the majority of `cherry-menu`'s functionality explained. It's not rocket science, just a quick hack of a shell-script, but it gets the job done pretty well.

There's one more important feature I didn't cover: if the `patch-id` changes during cherry-picking, `cherry-menu` will notice, and automatically add the old commit to the blacklist:

```
Cherry-pick / blacklist / skip 81204d4020, or quit ? c
```

```
[stable/grizzly 65cd526] Sync rpc from oslo-incubator.
```

```
Author: Russell Bryant <rbryant@CENSORED.com>
```

```
4 files changed, 47 insertions(+), 21 deletions(-)
```

```
The git patch-id changed during cherry-picking. This is normal when
the diff context changes or merge conflicts are resolved.
```

```
Blacklisted 81204d4020 so future runs won't attempt to duplicate the upstreami:
```

git rnotes – sharing the fruits of your labours

Maybe our project manager comes over and says “hey, we need those features backported to stable ASAP – I've asked Joe and Fred to help you out, OK?”

So far, all the work has happened in a local repository. Of course we can push our updated `stable/grizzly` branch to a public repository somewhere, so that Joe and Fred can see what we already cherry-picked. But they'd be still missing some valuable meta-data, i.e. the git notes we created which mark which commits to exclude from porting, and why. Unfortunately, git doesn't yet natively provide a convenient way for

pushing/pulling notes between repositories, because [the `refs/*` namespace hasn't been sufficiently standardized yet](#). To work around that, I wrote [git-rnotes](#) to avoid having to remember a slightly arcane sequence of git commands.

Usage is very simple, and mimics `git fetch/push/merge/pull`:

```
Usage: git-rnotes [options] SUBCOMMAND REMOTE
```

Options:

```
-h, --help      Show this help and exit
```

Subcommands:

```
fetch
```

```
push
```

```
merge
```

```
pull
```

For example, if you have a remote called `public`:

```
export GIT_NOTES_REF=refs/notes/upstreaming
git rnotes push public
```

or if Joe has updated his notes and pushed them to `public` whilst you also updated yours, simply do a `pull`, which is equivalent to a `fetch` followed by a `merge`:



```
export GIT_NOTES_REF=refs/notes/upstreaming
git rnotes pull public
```

Wrapping up

Sorry that was so long. But hopefully you agree it's relatively simple once you try it out. I'd love to know if anyone finds this useful – if you do, please leave a comment below! Of course pull requests against [my git-config repository](#) are always welcome. There are many other git-related hacks in that repository which you may also find useful.

Thanks for reading!



 [front_page](#), [geek](#), [hacks](#), [work](#) |  [backporting](#), [development](#), [forward-porting](#), [Free Software](#), [git](#), [hacking](#), [OpenStack](#), [software](#), [upstreaming](#), [version control](#)

15 Responses to “Easier upstreaming / back-porting of patch series with git”

1.  [Pádraig Brady](#) says:

[September 22, 2013 at 2:38 pm](#)

Oh some great stuff in here. This is a common and frustrating task. I was doing something more manual leveraging commit tags. Of course the “upstream first” mantra is useful here to minimize divergences.
Thank for taking the time to document this.

[Reply](#)

2.  [Judd Maltin](#) says:

[October 1, 2013 at 11:50 pm](#)

Adam, excellent writeup of your tool that I'll be trying out tonight! Thanks!

[Reply](#)

3. [OpenStack社区周报\(9.27 – 10.4\) | UnitedStack Inc.](#) says:

[October 11, 2013 at 4:11 am](#)

[...] [Git技巧之选择性提交](#) [...]

[Reply](#)

4.  [Ning](#) says:

[February 24, 2014 at 9:35 am](#)

The tools are very convenient for back porting patches. I've a problem here. When I was trying git-cherry-menu, the first commit 348cfedfca can be cherry picked successfully. But in your case, the command failed. There is definitely something wrong here. Can you please double check?

[Reply](#)

-  [Adam](#) says:

[February 24, 2014 at 10:37 am](#)

That's strange. Any chance you could paste a [gist](#) of your entire CLI session so I can see the details?

[Reply](#)

5.  [Ning](#) says:

[February 25, 2014 at 5:47 am](#)

Here is the gist log. I just copied the commands you gave in this article.

<https://gist.github.com/ningjiang/cca14f9b4b76bb1c8a66>

My git version is 1.7.9.5

[Reply](#)

6.  [Adam](#) says:

[February 25, 2014 at 3:41 pm](#)

That's very weird – I can reproduce this now. I can't explain it, but hopefully it won't prevent you from following through the rest of the example and learning how my tools work?

[Reply](#)

7.  [Ning](#) says:

[February 25, 2014 at 5:11 pm](#)

Thanks for the confirmation. I was able to follow all the examples. It just confused me for a while.

I have a suggestion after trying the tools.

Regarding this feature "if the patch-id changes during cherry-picking, cherry-menu will notice, and automatically add the old commit to the blacklist", it uses "skip: all" in current implementation. This will prevent the commit to be ported to

any other branches later. I think the better way is to just skip the branch which you specify as the command argument. What do you think of it?

[Reply](#)

8.  [Adam](#) says:

[February 25, 2014 at 6:45 pm](#)

Yes that would be better, but unfortunately it requires git cherry-menu to be able to understand the (arguments of the) command it wraps, and I deliberately avoided that in order to keep it as flexible as possible by being able to wrap any command which has the right output format. So I'm not sure how best to do this. Suggestions (or even pull requests) very welcome!

[Reply](#)

9.  [Ning](#) says:

[February 26, 2014 at 9:40 am](#)

I got your point. Can we use "git notes edit" in this case? User can get a chance to modify the skip list.

[Reply](#)

10.  [Adam](#) says:

[February 26, 2014 at 11:56 am](#)

Sure, you can do that any time 😊

[Reply](#)

11.  [Ning](#) says:

[February 27, 2014 at 11:48 am](#)

Send a pull request to roughly implement the idea. Please help to review and comment.

[Reply](#)

12.  [Alex](#) says:

[August 12, 2021 at 4:26 pm](#)

Hi,

when I run the command:

```
git icing -v3
```

I get the following error:

Traceback (most recent call last):

```
11: from /home/user/git-config/bin/git-icing:287:in `'  
10: from /home/user/git-config/bin/git-icing:281:in `main'  
9: from /home/user/git-config/bin/git-icing:194:in `cherry'  
8: from /home/user/git-config/bin/git-icing:194:in `popen'  
7: from /home/user/git-config/bin/git-icing:195:in `block in cherry'  
6: from /home/user/git-config/bin/git-icing:195:in `each_line'  
5: from /home/user/git-config/bin/git-icing:197:in `block (2 levels) in cherry'  
4: from /home/user/git-config/bin/git-icing:137:in `classify_line'  
3: from /home/user/git-config/bin/git-icing:74:in `upstreaming_note'  
2: from /usr/lib/ruby/2.7.0/open3.rb:101:in `popen3'  
1: from /usr/lib/ruby/2.7.0/open3.rb:219:in `popen_run'  
/home/user/git-config/bin/git-icing:80:in `block in upstreaming_note': git notes  
-ref\\=upstreaming show 8fb6a57bf826ead71d154a1f846554ca exited with  
status 1: error: no note found for object  
8fb6a57bf826ead71d154a1f846554ca23843ea5. (RuntimeError)
```

Do you know what is going wrong?

Thanks in advance.

Regards,

Alexander

[Reply](#)

○  Alex says:

[August 12, 2021 at 4:49 pm](#)

the formatting deleted some part of my previous message. The command used was `git icing -v3 TARGET_BRANCH Start_Commit_ID`

[Reply](#)

■  [Adam](#) says:

[August 12, 2021 at 5:06 pm](#)

Thanks for the report, any chance you could file it at <https://github.com/aspiers/git-config/issues/new>? Much easier to deal with technical issues there than on this blog post!

[Reply](#)

Leave a Reply

Name (required)

Mail (will not be published) (required)

Web site

☐ I'm not a robot

reCAPTCHA
[Privacy](#) - [Terms](#)

Add your Reply

← [new music video: acoustic version of Jóga by Björk, with Emma Smith](#)
[more uses for git notes, and hidden treasures in Gerrit](#) →

• Archives

- [► 2019 \(2\)](#)
- [► 2018 \(3\)](#)
- [► 2017 \(3\)](#)
- [► 2015 \(4\)](#)

- [▶2014 \(5\)](#)
- [▼2013 \(9\)](#)
 - [▶November\(1\)](#)
 - [▶October\(1\)](#)
 - [▼September\(2\)](#)
 - [Easier upstreaming / back-porting of patch series with git](#)
 - [new music video: acoustic version of Jóga by Björk, with Emma Smith](#)
 - [▶August\(1\)](#)
 - [▶May\(1\)](#)
 - [▶February\(2\)](#)
 - [▶January\(1\)](#)
- [▶2012 \(2\)](#)
- [▶2011 \(6\)](#)
- [▶2010 \(6\)](#)
- [▶2009 \(16\)](#)


• Categories

- [front page](#) (53)
- [fun](#) (3)
- [geek](#) (49)
 - [hacks](#) (13)
- [hidden](#) (24)
- [music](#) (14)
- [training](#) (11)
- [travel](#) (8)
- [work](#) (23)

• Tags

[android](#) [ant](#) [architecture](#) [Asia](#) [blog](#) [blogging](#) [career](#) [cello](#) [cloud](#) [CompuTrainer](#) [cycling](#) [dependencies](#)
[development](#) [Free Software](#) [gigs](#) [git](#) [github](#) [gtd](#) [hacking](#) [java](#) [javascript](#) [jazz](#) [KDE](#) [kml](#) [lifehacks](#)
[Linux](#) [make](#) [mobile](#) [mymaps](#) [mytracks](#) [obs](#) [OpenStack](#) [openSUSE](#) [phone](#) [rants](#) [rotc](#) [routes](#) [software](#) [suse](#)
[triathlon](#) [ubuntu](#) [usability](#) [version control](#) [VMware](#) [xml](#)

Structured Procrastination is powered by [WordPress](#)

Panorama Theme by  [Themocracy](#)