Collaborative Policymaking

How technology is making it easier to work together than alone

@BenBalter (http://twitter.com/benbalter)

government@github.com (mailto:government@github.com)

ben.balter.com/collaborative-policymaking (http://ben.balter.com/collaborative-policymaking)

Ye Olde Workflow

Hello team -

Latest version of the document attached.
Please submit edit directly to me by COB tomorrow.

...

---

revised-final-final-revised-final-bens-edits-20130401.docx
1K View Download

⏪ The Olden Days

Creation distinct from publication

High barrier to collaboration

Everyone for themselves

Share as late as possible, if at all

Us/them dichotomy, outsiders as liability

What’s a wheel? :see_no_evil:

⚙ Bespoke Collaboration

Discussions happen in private (:e-mail:, hallway)

Changes made opaquely, manually

Organization knowledge stored in people

Low fidelity (memory, :bus: factor, :watch:)

:calendar:, blockers as a sunk cost

A Brief History Of Open Source

Let's Say You're A Lawyer...

Impotent Legal Document

Whereas the parties agree to as follows:

0.

Heretofor this agreement is made and entered into as 17 September 2013 (“Effective Date”), by and between Disclosing Party Name, (“the Disclosing Party”) and Recipient Name, (“the Recipient”) (collectively, “the Parties”).

1.

Notwithstanding, for purposes of this Agreement, “Confidential Information” shall mean any and all non-public information, including, without limitation, technical, developmental, marketing, sales, operating, performance, cost, know-how, business plans, business methods, and process information, disclosed to the Recipient…

~~Impotent~~ Important Legal Document

Whereas the parties agree to as follows:

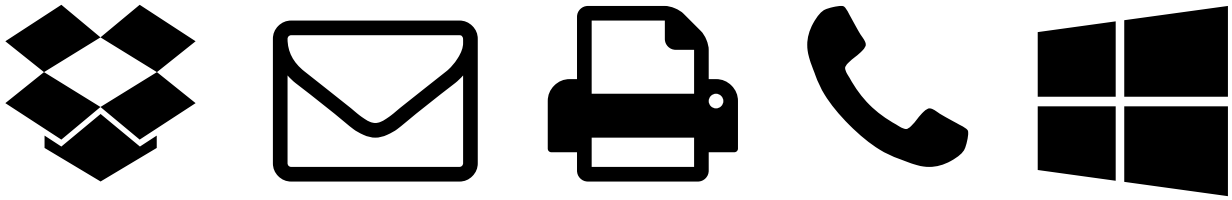
0.

Heretofor this agreement is made and entered into as 17 September 2013 (“Effective Date”), by and between Disclosing Party Name, (“the Disclosing Party”) and Recipient Name, (“the Recipient”) (collectively, “the Parties”).

1.

Notwithstanding, for purposes of this Agreement, “Confidential Information” shall mean any and all non-public information, including, without limitation, technical, developmental, marketing, sales, operating, performance, cost, know-how, business plans, business methods, and process information, disclosed to the Recipient…

Then What?



1985 To Now

Server
Publish
Contribute
→
Republi
↓
↑
You
Consume
→
Modify
→
Patch

The Contribution Model

Inconsistent

Error prone

Complex

Doesn’t scale

How To Contribute To The Linux Kernel

```
                    How to Get Your Change Into the Linux Kernel
                                   or
                    Care And Operation Of Your Linux Turvalds


    For a person or company who wishes to submit a change to the Linux
    kernel, the process can sometimes be daunting if you're not familiar
    with "the system." This text is a collection of suggestions which
    can greatly increase the chances of your change being accepted.

    Read Documentation/SubmitChecklist for a list of items to check
    before submitting code. If you are submitting a driver, also read
    Documentation/SubmittingDrivers.


                    ------------------------------------------
                    SECTION 1 - CREATING AND SENDING YOUR CHANGE
                    ------------------------------------------


                        1) "diff -up"
                        -------------

                    Use "diff -up" or "diff -uprN" to create patches.

    All changes to the Linux kernel occur in the form of patches, as
    generated by diff(1). When creating your patch, make sure to create it
    in "unified diff" format, as supplied by the "-u" argument to diff(1).
    Also, please use the "-p" argument which shows which C function each
    change is in - that makes the resultant diff a lot easier to read.
    Patches should be based in the root kernel source directory,
                    not in any lower subdirectory.

    To create a patch for a single file, it is often sufficient to do:

                    SECTEND= linux-2.4
                    MYFILE= drivers/net/eg/driver.c

                    cd SECTEND
                    cp SMYFILE SMYFILE.orig
                    vi SMYFILE      # make your change
                    cd ..
                    diff -up SECTEND/SMYFILE.orig.j > /tmp/patch

    To create a patch for multiple files, you should unpack a "vanilla",
    or unmodified kernel source tree, and generate a diff against your
                    own source tree. For example:

                    MYDEC= /devel/linux-2.4

                    tar xvfz linux-2.4.12.tar.gz
                    mv linux-2.4.12 linux-2.4.12-vanilla
                    diff -uprN -X linux-2.4.12-vanilla/Documentation/dontdiff \
                    linux-2.4.12-vanilla SMYDEC > /tmp/patch

    "dontdiff" is a list of files which are generated by the kernel during
    the build process, and should be ignored in any diff(1)-generated
    patch. The "dontdiff" file is included in the kernel tree in
                    2.4.12 and later.

    Make sure your patch does not include any extra files which do not
    belong in a patch submission. Make sure to review your patch -after-
                    generated it with diff(1), to ensure accuracy.

    If your changes produce a lot of deltas, you may want to look into
    splitting them into individual patches which modify things in
    logical stages. This will facilitate easier reviewing by other
    kernel developers, very important if you want your patch accepted.
                    There are a number of scripts which can aid in this:

                                    quilt:
                    http://savannah.nongnu.org/projects/quilt

                    Andrew Morton's patch scripts:
                    http://ozerkin.kernel.org/~akpm/stuff/patch-scripts.tar.gz
    Instead of these scripts, quilt is the recommended patch management
                    tool (see above).


                    2) Describe your changes.

    Describe the technical detail of the change(s) your patch includes.

    Be as specific as possible. The WORST descriptions possible include
    things like "update driver X", "bug fix for driver X", or "this patch
                    includes updates for subsystem X. Please apply."

    The maintainer will thank you if you write your patch description in a
    form which can be easily pulled into Linux's source code management
                    system, git, as a "commit log". See #15, below.

    If your description starts to get long, that's a sign that you probably
                    need to split up your patch. See #3, next.

    When you submit or resubmit a patch or patch series, include the
    complete patch description and justification for it. Don't just
    say that this is version N of the patch (series). Don't expect the
    patch merger to refer back to earlier patch versions or referenced
    URLs to find the patch description and put that into the patch.
    I.e., the patch (series) and its description should be self-contained.
    This benefits both the patch merger(s) and reviewers. Some reviewers
                    probably didn't even receive earlier versions of the patch.

    If the patch fixes a logged bug entry, refer to that bug entry by
                    number and URL.


                    3) Separate your changes.

                    Separate logical changes into a single patch file.

    For example, if your changes include both bug fixes and performance
    enhancements for a single driver, separate those changes into two
    or more patches. If your changes include an API update, and a new
    driver which uses that new API, separate those into two patches.

    On the other hand, if you make a single change to numerous files,
    group those changes into a single patch. Thus a single logical change
                    is contained within a single patch.

    If one patch depends on another patch in order for a change to be
    complete, that is OK. Simply note "this patch depends on patch X"
                    in your patch description.

    If you cannot condense your patch set into a smaller set of patches,
    then only post say 15 or so at a time and wait for review and integration.


                    4) Style check your changes.

    Check your patch for basic style violations, details of which can be
    found in Documentation/CodingStyle. Failure to do so simply wastes
                    the reviewers time and will get your patch rejected, probably
                    without even being read.

    At a minimum you should check your patches with the patch style
                    checker prior to submission (scripts/checkpatch.pl). You should
```

be able to justify all violations that remain in your patch.

5) Select e-mail destination.

Look through the MAINTAINERS file and the source code, and determine if your change applies to a specific subsystem of the kernel, with an assigned maintainer. If so, e-mail that person. The script scripts/get_maintainer.pl can be very useful at this step.

If no maintainer is listed, or the maintainer does not respond, send your patch to the primary Linux kernel developer's mailing list, linux-kernel@vger.kernel.org. Most kernel developers monitor this e-mail list, and can comment on your changes.

Do not send more than 15 patches at once to the vger mailing lists!!!

Linus Torvalds is the final arbiter of all changes accepted into the Linux kernel. His e-mail address is <torvalds@linux-foundation.org>. He gets a lot of e-mail, so typically you should do your best to -avoid- sending him e-mail.

Patches which are bug fixes, are "obvious" changes, or similarly require little discussion should be sent or CC'd to Linus. Patches which require discussion or do not have a clear advantage should usually be sent first to linux-kernel. Only after the patch is discussed should the patch then be submitted to Linus.

6) Select your CC (e-mail carbon copy) list.

Unless you have a reason NOT to do so, CC linux-kernel@vger.kernel.org.

Other kernel developers besides Linus need to be aware of your change, so that they may comment on it and offer code review and suggestions. linux-kernel is the primary Linux kernel developer mailing list. Other mailing lists are available for specific subsystems, such as USB, framebuffer devices, the VFS, the SCSI subsystem, etc. See the MAINTAINERS file for a mailing list that relates specifically to your change.

Majordomo lists of VGER.KERNEL.ORG at:
<http://vger.kernel.org/vger-lists.html>

If changes affect userland-kernel interfaces, please send the MAN-PAGES maintainer (as listed in the MAINTAINERS file) a man-pages patch, or at least a notification of the change, so that some information makes it into the manual pages.

Even if the maintainer did not respond in step #5, make sure to ALWAYS copy the maintainer when you change their code.

For small patches you may want to CC the Trivial Patch Monkey trivial@kernel.org which collects "trivial" patches. Have a look into the MAINTAINERS file for its current manager.
Trivial patches must qualify for one of the following rules:
  Spelling fixes in documentation
  Spelling fixes which could break grep(1)
  Warning fixes (cluttering with useless warnings is bad)
  Compilation fixes (only if they are actually correct)
  Runtime fixes (only if they actually fix things)
  Removing use of deprecated functions/macros (eg. check_region)
  Contact detail and documentation fixes
  Non-portable code replaced by portable code (even in arch-specific, since people copy, as long as it's trivial)
  Any fix by the author/maintainer of the file (ie. patch monkey in re-transmission mode)

7) No MIME, no links, no compression, no attachments. Just plain text.

Linus and other kernel developers need to be able to read and comment on the changes you are submitting. It is important for a kernel developer to be able to "quote" your changes, using standard e-mail tools, so that they may comment on specific portions of your code.

For this reason, all patches should be submitting e-mail "inline".
WARNING: Be wary of your editor's word-wrap corrupting your patch, if you choose to cut-n-paste your patch.

Do not attach the patch as a MIME attachment, compressed or not. Many popular e-mail applications will not always transmit a MIME attachment as plain text, making it impossible to comment on your code. A MIME attachment also takes Linus a bit more time to process, decreasing the likelihood of your MIME-attached change being accepted.

Exception: If your mailer is mangling patches then someone may ask you to re-send them using MIME.

See Documentation/email-clients.txt for hints about configuring your e-mail client so that it sends your patches untouched.

8) E-mail size.

When sending patches to Linus, always follow step #7.

Large changes are not appropriate for mailing lists, and some maintainers. If your patch, uncompressed, exceeds 300 kB in size, it is preferred that you store your patch on an Internet-accessible server, and provide instead a URL (link) pointing to your patch.

9) Name your kernel version.

It is important to note, either in the subject line or in the patch description, the kernel version to which this patch applies.

If the patch does not apply cleanly to the latest kernel version, Linus will not apply it.

10) Don't get discouraged. Re-submit.

After you have submitted your change, be patient and wait. If Linus likes your change and applies it, it will appear in the next version of the kernel that he releases.

However, if your change doesn't appear in the next version of the kernel, there could be any number of reasons. It's YOUR job to narrow down those reasons, correct what was wrong, and submit your updated change.

It is quite common for Linus to "drop" your patch without comment, that's the nature of the system. If he drops your patch, it could be due to
* Your patch did not apply cleanly to the latest kernel version.
* Your patch was not sufficiently discussed on linux-kernel.
  * A style issue (see section 2).
  * An e-mail formatting issue (re-read this section).
  * A technical problem with your change.
* He gets tons of e-mail, and yours got lost in the shuffle.
  * You are being annoying.

When in doubt, solicit comments on linux-kernel mailing list.

11) Include PATCH in the subject

Due to high e-mail traffic to Linus, and to linux-kernel, it is common convention to prefix your subject line with [PATCH]. This lets Linus and other kernel developers more easily distinguish patches from other e-mail discussions.

12) Sign your work

To improve tracking of who did what, especially with patches that can percolate to their final resting place in the kernel through several layers of maintainers, we've introduced a "sign-off" procedure on patches that are being emailed around.

The sign-off is a simple line at the end of the explanation for the patch, which certifies that you wrote it or otherwise have the right to pass it on as an open-source patch. The rules are pretty simple: if you can certify the below:

Developer's Certificate of Origin 1.1

By making a contribution to this project, I certify that:

(a) The contribution was created in whole or in part by me and I have the right to submit it under the open source license indicated in the file; or

(b) The contribution is based upon previous work that, to the best of my knowledge, is covered under an appropriate open source license and I have the right under that license to submit that work with modifications, whether created in whole or in part by me, under the same open source license (unless I am permitted to submit under a different license), as indicated in the file; or

(c) The contribution was provided directly to me by some other person who certified (a), (b) or (c) and I have not modified it.

(d) I understand and agree that this project and the contribution are public and that a record of the contribution (including all personal information I submit with it, including my sign-off) is maintained indefinitely and may be redistributed consistent with this project or the open source license(s) involved.

then you just add a line saying

Signed-off-by: Random J Developer <random@developer.example.org>

using your real name (sorry, no pseudonyms or anonymous contributions.)

Some people also put extra tags at the end. They'll just be ignored for now, but you can do this to mark internal company procedures or just point out some special detail about the sign-off.

If you are a subsystem or branch maintainer, sometimes you need to slightly modify patches you receive in order to merge them, because the code is not exactly the same in your tree and the submitters'. If you stick strictly to rule (c), you should ask the submitter to rediff, but this is a totally counter-productive waste of time and energy. Rule (b) allows you to adjust the code, but then it is very impolite to change one submitter's code and make him endorse your bugs. To solve this problem, it is recommended that you add a line between the last Signed-off-by header and yours, indicating the nature of your changes. While there is nothing mandatory about this, it seems like prepending the description with your mail and/or name, all enclosed in square brackets, is noticeable enough to make it obvious that you are responsible for last-minute changes. Example :

Signed-off-by: Random J Developer <random@developer.example.org>
[lucky@maintainer.example.org: struct foo moved from foo.c to foo.h]
Signed-off-by: Lucky K Maintainer <lucky@maintainer.example.org>

This practice is particularly helpful if you maintain a stable branch and want at the same time to credit the author, track changes, merge the fix, and protect the submitter from complaints. Note that under no circumstances can you change the author's identity (the From header), as it is the one which appears in the changelog.

Special note to back-porters: It seems to be a common and useful practice to insert an indication of the origin of a patch at the top of the commit message (just after the subject line) to facilitate tracking. For instance, here's what we see in 2.4-stable :

Date: Tue May 13 19:10:30 2008 +0000

ACPI: libiasl regression in 2.6.25: fix nop timer handling

commit 4cf1043593db6a337f10e09a2c24e4fc9d7f672 upstream

And here's what appears in 2.4 :

Date: Tue May 13 22:12:27 2008 +0200

wireless, airo: waitbusy() won't delay

[backport of 2.4 commit b7acbdfbd1f277c1eb23f344f899cfa4cd0bf36a]

Whatever the format, this information provides a valuable help to people tracking your trees, and to people trying to trouble-shoot bugs in your tree.

13) When to use Acked-by: and Cc:

The Signed-off-by: tag indicates that the signer was involved in the development of the patch, or that he/she was in the patch's delivery path.

If a person was not directly involved in the preparation or handling of a patch but wishes to signify and record their approval of it then they can arrange to have an Acked-by: line added to the patch's changelog.

Acked-by: is often used by the maintainer of the affected code when that maintainer neither contributed to nor forwarded the patch.

Acked-by: is not as formal as Signed-off-by:. It is a record that the acker has at least reviewed the patch and has indicated acceptance. Hence patch mergers will sometimes manually convert an acker's "yep, looks good to me" into an Acked-by:.

Acked-by: does not necessarily indicate acknowledgement of the entire patch. For example, if a patch affects multiple subsystems and has an Acked-by: from one subsystem maintainer then this usually indicates acknowledgement of just the part which affects that maintainer's code. Judgement should be used here. When in doubt people should refer to the original discussion in the mailing list archives.

If a person has had the opportunity to comment on a patch, but has not provided such comments, you may optionally add a "Cc:" tag to the patch. This is the only tag which might be added without an explicit action by the person it names. This tag documents that potentially interested parties have been included in the discussion

14) Using Reported-by:, Tested-by:, Reviewed-by: and Suggested-by:

If this patch fixes a problem reported by somebody else, consider adding a Reported-by: tag to credit the reporter for their contribution. Please note that this tag should not be added without the reporter's permission, especially if the problem was not reported in a public forum. That said, if we diligently credit our bug reporters, they will, hopefully, be inspired to help us again in the future.

A Tested-by: tag indicates that the patch has been successfully tested (in some environment) by the person named. This tag informs maintainers that some testing has been performed, provides a means to locate testers for future patches, and ensures credit for the testers.

Reviewed-by:, instead, indicates that the patch has been reviewed and found acceptable according to the Reviewer's Statement:

Reviewer's statement of oversight

By offering my Reviewed-by: tag, I state that:

 (a) I have carried out a technical review of this patch to evaluate its appropriateness and readiness for inclusion into the mainline kernel.

 (b) Any problems, concerns, or questions relating to the patch have been communicated back to the submitter. I am satisfied with the submitter's response to my comments.

 (c) While there may be things that could be improved with this submission, I believe that it is, at this time, (1) a worthwhile modification to the kernel, and (2) free of known issues which would argue against its inclusion.

 (d) While I have reviewed the patch and believe it to be sound, I do not (unless explicitly stated elsewhere) make any warranties or guarantees that it will achieve its stated purpose or function properly in any given situation.

A Reviewed-by tag is a statement of opinion that the patch is an appropriate modification of the kernel without any remaining serious technical issues. Any interested reviewer (who has done the work) can offer a Reviewed-by tag for a patch. This tag serves to give credit to reviewers and to inform maintainers of the degree of review which has been done on the patch. Reviewed-by: tags, when supplied by reviewers known to understand the subject area and to perform thorough reviews, will normally increase the likelihood of your patch getting into the kernel.

A Suggested-by: tag indicates that the patch idea is suggested by the person named and ensures credit to the person for the idea. Please note that this tag should not be added without the reporter's permission, especially if the idea was not posted in a public forum. That said, if we diligently credit our idea reporters, they will, hopefully, be inspired to help us again in the future.

15) The canonical patch format

The canonical patch subject line is:

Subject: [PATCH 001/123] subsystem: summary phrase

The canonical patch message body contains the following:

 - A "from" line specifying the patch author.

 - An empty line.

 - The body of the explanation, which will be copied to the permanent changelog to describe this patch.

 - The "Signed-off-by:" lines, described above, which will also go in the changelog.

 - A marker line containing simply "---".

 - Any additional comments not suitable for the changelog.

 - The actual patch (diff output).

The Subject line format makes it very easy to sort the emails alphabetically by subject line - pretty much any email reader will support that - since because the sequence number is zero-padded, the numerical and alphabetic sort is the same.

The "subsystem" in the email's Subject should identify which area or subsystem of the kernel is being patched.

The "summary phrase" in the email's Subject should concisely describe the patch which that email contains. The "summary phrase" should not be a filename. Do not use the same "summary phrase" for every patch in a whole patch series (where a "patch series" is an ordered sequence of multiple, related patches).

Bear in mind that the "summary phrase" of your email becomes a globally-unique identifier for that patch. It propagates all the way into the git changelog. The "summary phrase" may later be used in developer discussions which refer to the patch. People will want to google for the "summary phrase" to read discussion regarding that patch. It will also be the only thing that people may quickly see when, two or three months later, they are going through perhaps thousands of patches using tools such as "gitk" or "git log --oneline".

For these reasons, the "summary" must be no more than 70-75 characters, and it must describe both what the patch changes, as well as why the patch might be applied. It is challenging to be both succinct and descriptive, but that is what a well-written summary should do.

The "summary phrase" may be prefixed by tags enclosed in square brackets: "Subject: [PATCH tag] <summary phrase>". The tags are not considered part of the summary phrase, but describe how the patch should be treated. Common tags might include a version descriptor if the multiple versions of the patch have been sent out in response to comments (i.e., "v1, v2, v3"), or "RFC" to indicate a request for comments. If there are four patches in a patch series the individual patches may be numbered like this: 1/4, 2/4, 3/4, 4/4. This assures that developers understand the order in which the patches should be applied and that they have reviewed or applied all of the patches in the patch series.

A couple of example Subjects:

Subject: [patch 2/5] ext3: improve scalability of bitmap searching
Subject: [PATCHv2 001/207] x86: fix eflags tracking

The "from" line must be the very first line in the message body, and has the form:

From: Original Author <author@example.com>

The "from" line specifies who will be credited as the author of the patch in the permanent changelog. If the "from" line is missing, then the "From:" line from the email header will be used to determine the patch author in the changelog.

The explanation body will be committed to the permanent source changelog, so should make sense to a competent reader who has long since forgotten the immediate details of the discussion that might have led to this patch. Including symptoms of the failure which the patch addresses (kernel log messages, oops messages, etc.) is especially useful for people who might be searching the kernel logs looking for the applicable patch. If a patch fixes a compile failure, it may not be necessary to include _all_ of the compile failures; just enough that it is likely that someone searching for the patch can find it. As in the "summary phrase", it is important to be both succinct as well as descriptive.

The "---" marker line serves the essential purpose of marking for patch handling tools where the changelog message ends.

One good use for the additional comments after the "---" marker is for a diffstat, to show what files have changed, and the number of inserted and deleted lines per file. A diffstat is especially useful on bigger patches. Other comments relevant only to the moment or the maintainer, not suitable for the permanent changelog, should also go here. A good example of such comments might be "patch changelogs" which describe what has changed between the v1 and v2 version of the patch.

If you are going to include a diffstat after the "---" marker, please use diffstat options "-p 1 -w 70" so that filenames are listed from the top of the kernel source tree and don't use too much horizontal space (easily fit in 80 columns, maybe with some indentation).

See more details on the proper patch format in the following references.

16) Sending "git pull" requests (from Linus emails)

Please write the git repo address and branch name alone on the same line so that I can't even by mistake pull from the wrong branch, and so that a triple-click just selects the whole thing.

So the proper format is something along the lines of:

"Please pull from

git://jdelvare.pck.nerim.net/jdelvare-2.6 i2c-for-linus

to get these changes:"

so that I don't have to hunt-and-peck for the address and inevitably
get it wrong (actually, I've only gotten it wrong a few times, and
checking against the diffstat tells me when I get it wrong, but I'm
just a not more comfortable when I don't have to 'look for' the right
thing to pull, and double-check that I have the right branch-name).

Please use "git diff -M --stat --summary" to generate the diffstat:
the -M enables rename detection, and the summary enables a summary of
new/deleted or renamed files.

With rename detection, the statistics are rather different [...]
because git will notice that a fair number of the changes are renames.

SECTION 2 - HOWTO, TIPS, AND TRICKS

This section lists many of the common "rules" associated with code
submitted to the kernel. There are always exceptions... but you must
have a really good reason for doing so. You could probably call this
section Linus Computer Science 101.

1) Read Documentation/CodingStyle

Nuff said. If your code deviates too much from this, it is likely
to be rejected without further review, and without comment.

One significant exception is when moving code from one file to
another -- in this case you should not modify the moved code at all in
the same patch which moves it. This clearly delineates the act of
moving the code and your changes. This greatly aids review of the
actual differences and allows tools to better track the history of
the code itself.

Check your patches with the patch style checker prior to submission
(scripts/checkpatch.pl). The style checker should be viewed as
a guide not as the final word. If your code looks better with
a violation then its probably best left alone.

The checker reports at three levels:
- ERROR: things that are very likely to be wrong
- WARNING: things requiring careful review
- CHECK: things requiring thought

You should be able to justify all violations that remain in your
patch.

2) #ifdefs are ugly

Code cluttered with ifdefs is difficult to read and maintain. Don't do
it. Instead, put your ifdefs in a header, and conditionally define
'static inline' functions, or macros, which are used in the code.
Let the compiler optimize away the "no-op" case.

Simple example, of poor code:

dev = alloc_etherdev (sizeof(struct funky_private));
if (!dev)
return -ENODEV;
#ifdef CONFIG_NET_FUNKINESS
init_funky_net(dev);
#endif

Cleaned-up example:

(in header)
#ifndef CONFIG_NET_FUNKINESS
static inline void init_funky_net (struct net_device *d) {}
#endif

(in the code itself)
dev = alloc_etherdev (sizeof(struct funky_private));
if (!dev)
return -ENODEV;
init_funky_net(dev);

3) 'static inline' is better than a macro

Static inline functions are greatly preferred over macros.
They provide type safety, have no length limitations, no formatting
limitations, and under gcc they are as cheap as macros.

Macros should only be used for cases where a static inline is clearly
suboptimal [there are a few, isolated cases of this in fast paths],
or where it is impossible to use a static inline function [such as
string-izing].

'static inline' is preferred over 'static __inline__', 'extern inline',
and 'extern __inline__'.

4) Don't over-design.

Don't try to anticipate nebulous future cases which may or may not
be useful: "make it as simple as you can, and no simpler."

SECTION 3 - REFERENCES

Andrew Morton, "The perfect patch" (tpp).
<http://userweb.kernel.org/~akpm/stuff/tpp.txt>

Jeff Garzik, "Linux kernel patch submission format".
<http://linux.yyz.us/patch-format.html>

Greg Kroah-Hartman, "How to piss off a kernel subsystem maintainer".
<http://www.kroah.com/log/linux/maintainer.html>
<http://www.kroah.com/log/linux/maintainer-02.html>
<http://www.kroah.com/log/linux/maintainer-03.html>
<http://www.kroah.com/log/linux/maintainer-04.html>
<http://www.kroah.com/log/linux/maintainer-05.html>

NO!!!! No more huge patch bombs to linux-kernel@vger.kernel.org people!
<http://marc.theaimsgroup.com/?l=linux-kernel&m=112112749912944&w=2>

Kernel Documentation/CodingStyle:
<http://sosdg.org/~coywolf/lxr/source/Documentation/CodingStyle>

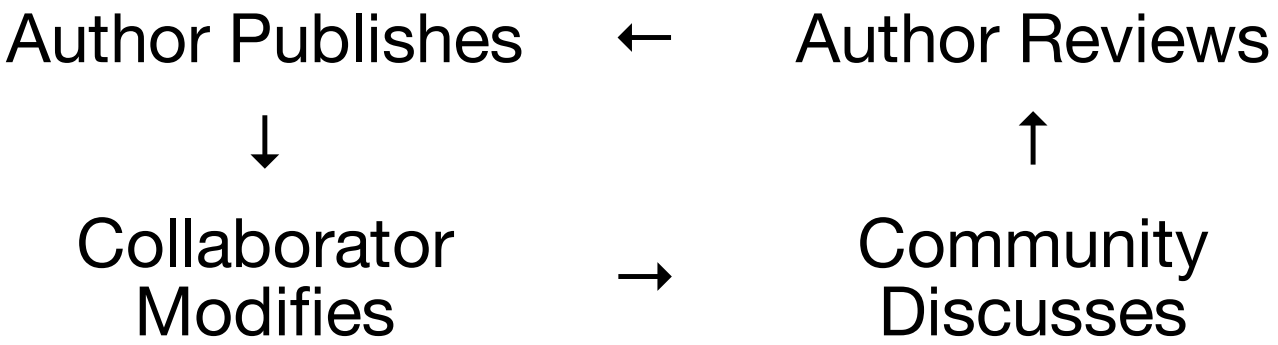
Linus Torvalds's mail on the canonical patch format:
<http://lkml.org/lkml/2005/4/7/183>

Andi Kleen, "On submitting kernel patches"
Some strategies to get difficult or controversial changes in.
http://halobates.de/on-submitting-patches.pdf

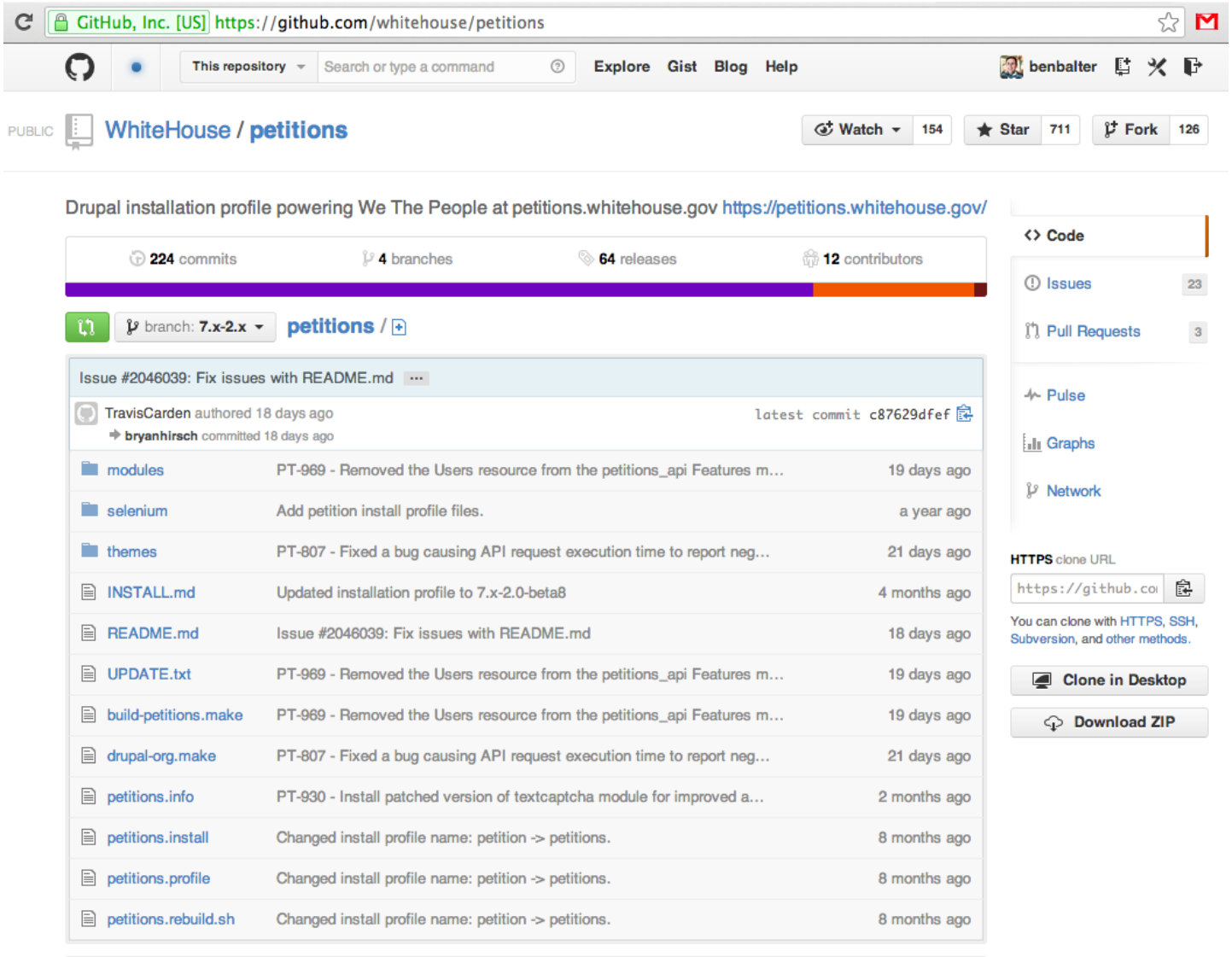
--

How Open Source Has Changed

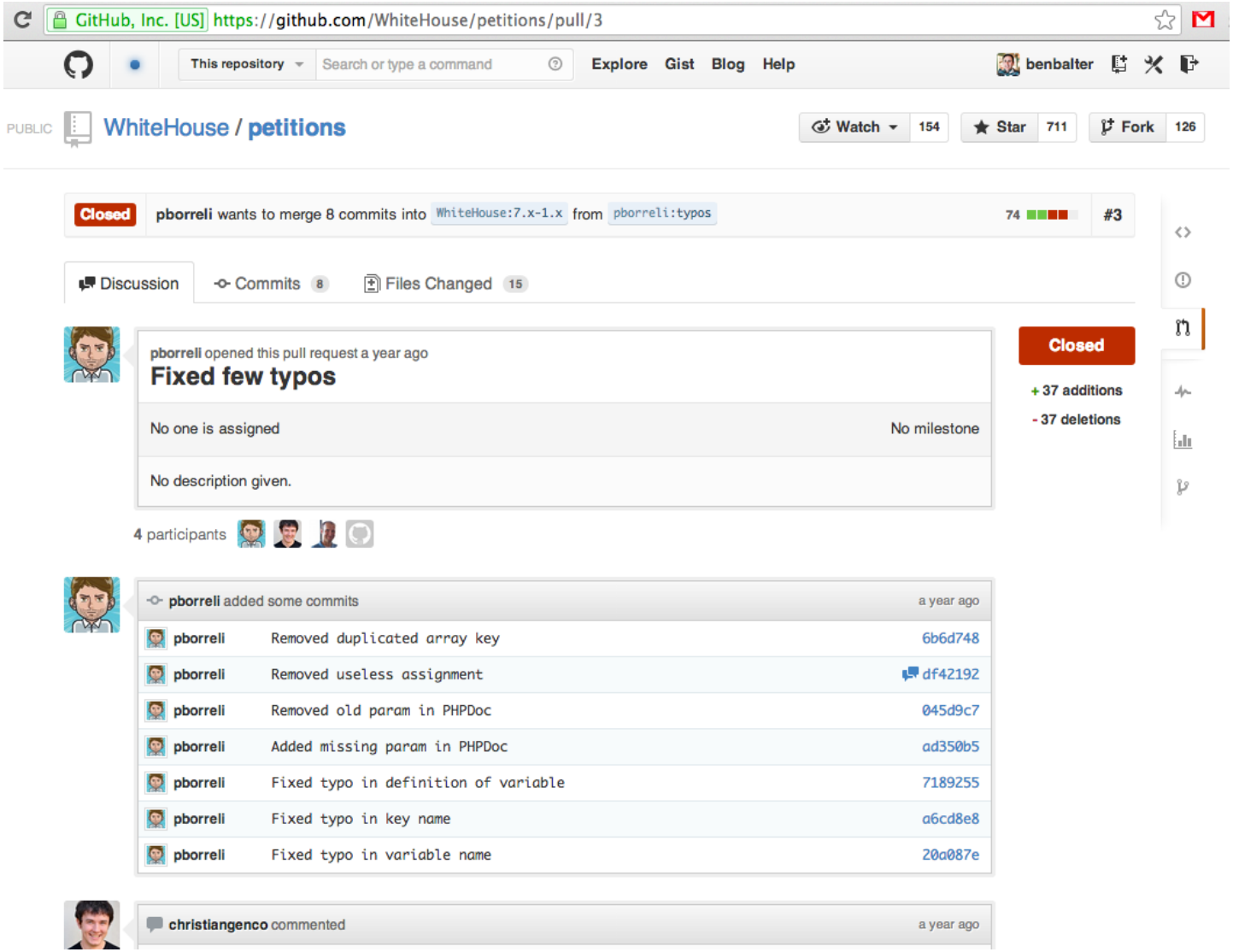
Open Source Today



Author Publishes



Collaborator Modifies



Community Discusses

christiangenco commented on `df42192` a year ago

Hah.

pborreli referenced this pull request a year ago

Pull Request #7: Fixed typo in method (BC BREAK) **Closed**

greggles commented a year ago

The changes to govdelivery should be provided as a patch to http://drupal.org/project/govdelivery

pborreli added a commit a year ago

pborreli `Fixed typo` ··· d9ae688

pborreli commented a year ago

@greggles thanks, i removed all the changes related to govdelivery.

lheyman commented a year ago

Thanks **@pborreli**

We cherry-picked your typo commits: `ad350b5`, `df42192`, `a6cd8e8`.

As **@greggles** mentioned, the govdelivery updates should be contributed on the govdelivery project page. We had to fork our own module a little, but we will take your commits under advisement when we update the module on drupal.org

[EDIT: changed commit SHA]

Closed **lheyman** closed the pull request a year ago

Author Reviews

8 modules/custom/wh_petition_reporting/wh_petition_reporting.hourly.inc View file @ d9ae688

```
...  ...  @@ -21,7 +21,7 @@ class HourlyReportManager {
21   21       *
22   22       * Checks time of last build and sets indicator if new build is needed
23   23       *
24       -    * @todo: figour out what the default variable should be
     24   +    * @todo: figure out what the default variable should be
25   25       */
26   26      function __construct($debug = FALSE) {
27   27        $this->debug = $debug;
...  ...  @@ -140,7 +140,7 @@ class HourlyReportManager {
140  140     }
141  141
142  142     /**
143      -    * update the Drupal var and class attribte marking the last hour for which this ran
     143  +    * update the Drupal var and class attribute marking the last hour for which this ran
144  144      */
145  145     private function updateLastRun($lastRunTS) {
146  146       variable_set('wh_petition_reporting_hourly_last_run_hour', $lastRunTS);
...  ...  @@ -148,7 +148,7 @@ class HourlyReportManager {
148  148     }
149  149
150  150     /**
151      -    *  Wrapper to handle retireving data for an hour and writing it to the mongo table
     151  +    *  Wrapper to handle retrieving data for an hour and writing it to the mongo table
152  152      */
153  153     public function processHourRecord($hour_start_timestamp, $hour_end_timestamp) {
154  154       HourlyReportManager::writeHourRecord(HourlyReportManager::getHourRecord($hour_start_timestamp, $hou
...  ...  @@ -162,7 +162,7 @@ class HourlyReportManager {
162  162      * @param $hour_end_timestamp int
163  163      *   Unix timestamp for when the hour to get should end
164  164      * @param $delta_only boolean
165      -    *   Indicator of whether to get the total vlues or just the new records in that hour
     165  +    *   Indicator of whether to get the total values or just the new records in that hour
166  166      *
167  167      * @return array
168  168      */
```

The Collaboration Model

Simple

Open

Standardized

Scales

How To Contribute To Petitions.whitehouse.gov

Contributing

Anyone is encouraged to contribute to the project by forking (https://help.github.com/articles/fork-a-repo) and submitting a pull request.
(If you are new to GitHub, you might start with a basic tutorial (https://help.github.com/articles/set-up-git).)

The White House Open Data Policy

PROJECT OPEN DATA

Open Data Policy — Managing Information as an Asset

1. Background

Data is a valuable national resource and a strategic asset to the U.S. Government, its partners, and the public. Managing this data as an asset and making it available, discoverable, and usable – in a word, open – not only strengthens our democracy and promotes efficiency and effectiveness in government, but also has the potential to create economic opportunity and improve citizens' quality of life.

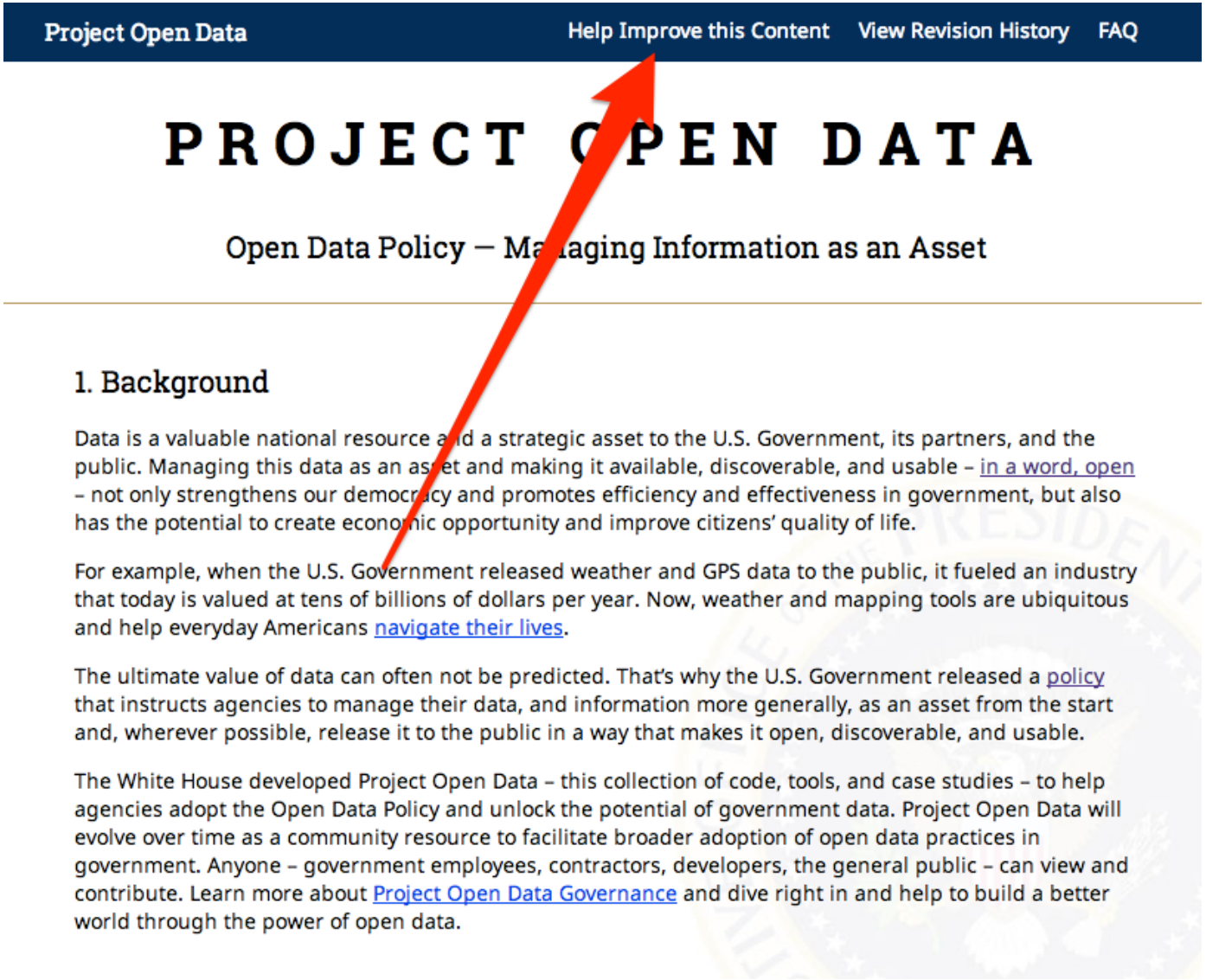
For example, when the U.S. Government released weather and GPS data to the public, it fueled an industry that today is valued at tens of billions of dollars per year. Now, weather and mapping tools are ubiquitous and help everyday Americans navigate their lives.

The ultimate value of data can often not be predicted. That's why the U.S. Government released a policy that instructs agencies to manage their data, and information more generally, as an asset from the start and, wherever possible, release it to the public in a way that makes it open, discoverable, and usable.

The White House developed Project Open Data – this collection of code, tools, and case studies – to help agencies adopt the Open Data Policy and unlock the potential of government data. Project Open Data will evolve over time as a community resource to facilitate broader adoption of open data practices in government. Anyone – government employees, contractors, developers, the general public – can view and contribute. Learn more about Project Open Data Governance and dive right in and help to build a better world through the power of open data.

Markdown

```
Data is a valuable national resource and a stra
tegic asset to the U.S. Government, its partner
s, and the public.  Managing this data as an as
set and making it available, discoverable, and
usable — [in a word, open](principles/) — not o
nly strengthens our democracy and promotes effi
ciency and effectiveness in government, but als
o has the potential to create economic opportun
ity and improve citizens' quality of life.
```


Project Open Data
Help Improve this Content
View Revision History
FAQ
PROJECT OPEN DATA
Open Data Policy — Managing Information as an Asset
1. Background
Data is a valuable national resource and a strategic asset to the U.S. Government, its partners, and the public. Managing this data as an asset and making it available, discoverable, and usable – in a word, open – not only strengthens our democracy and promotes efficiency and effectiveness in government, but also has the potential to create economic opportunity and improve citizens' quality of life.
For example, when the U.S. Government released weather and GPS data to the public, it fueled an industry that today is valued at tens of billions of dollars per year. Now, weather and mapping tools are ubiquitous and help everyday Americans navigate their lives.
The ultimate value of data can often not be predicted. That's why the U.S. Government released a policy that instructs agencies to manage their data, and information more generally, as an asset from the start and, wherever possible, release it to the public in a way that makes it open, discoverable, and usable.
The White House developed Project Open Data – this collection of code, tools, and case studies – to help agencies adopt the Open Data Policy and unlock the potential of government data. Project Open Data will evolve over time as a community resource to facilitate broader adoption of open data practices in government. Anyone – government employees, contractors, developers, the general public – can view and contribute. Learn more about Project Open Data Governance and dive right in and help to build a better world through the power of open data.

prose.io

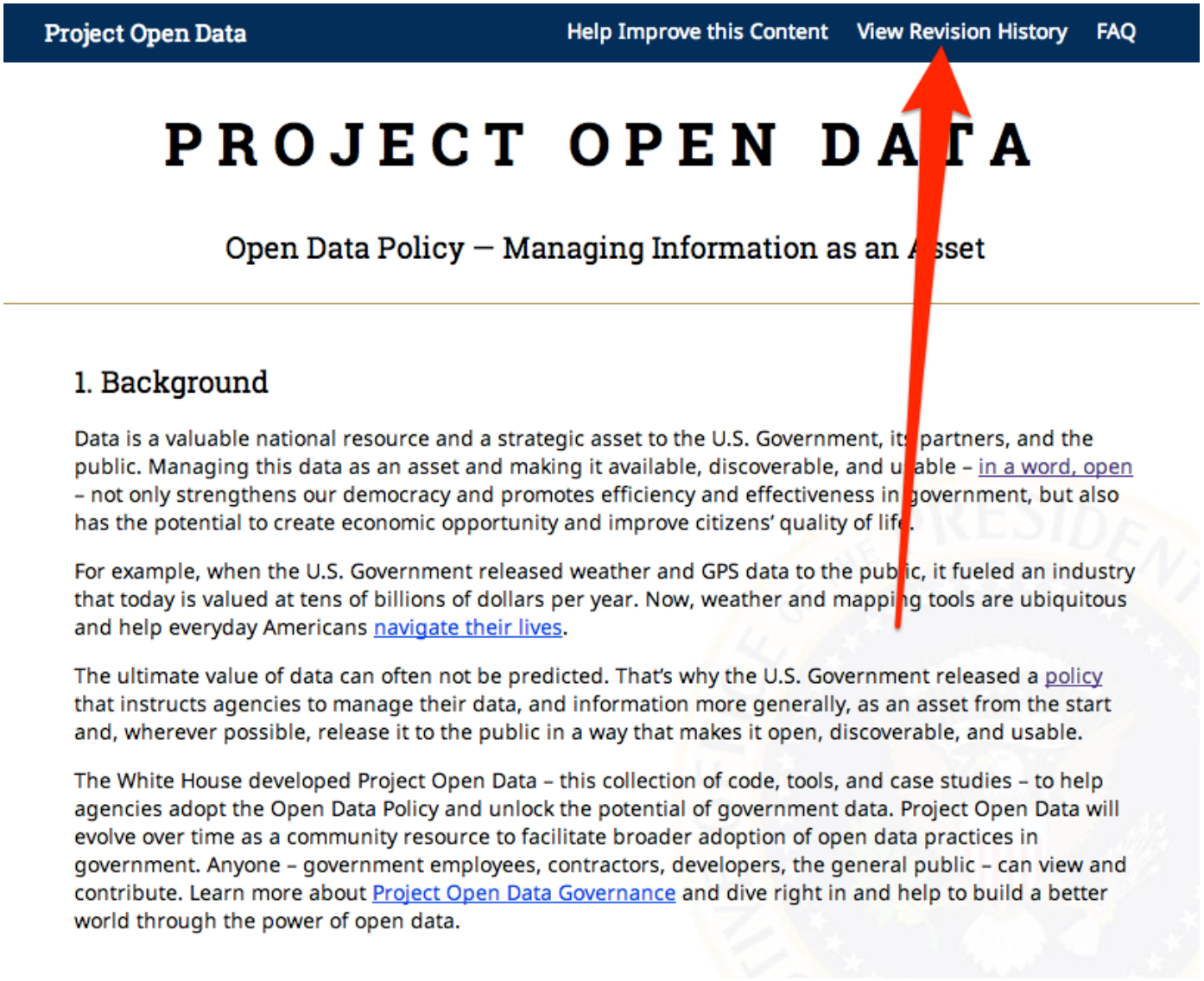
project-open-data / project-open-data.github.io

Project Open Data

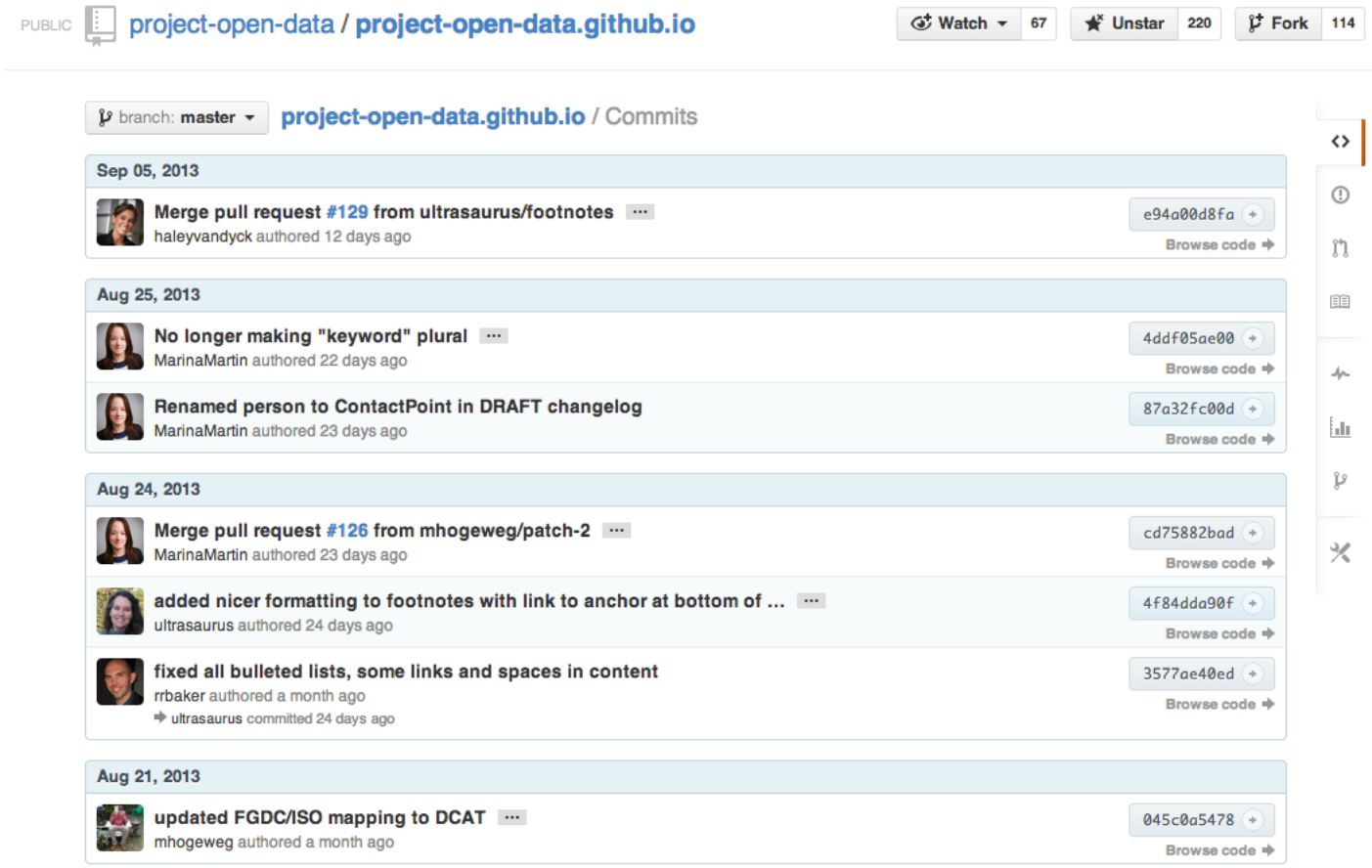
h2 h3 B Published

```
##1. Background

Data is a valuable national resource and a strategic asset to the U.S. Government,
its partners, and the public.  Managing this data as an asset and making it
available, discoverable, and usable — [in a word, open](principles/) — not only
strengthens our democracy and promotes efficiency and effectiveness in government,
but also has the potential to create economic opportunity and improve citizens'
quality of life.

For example, when the U.S. Government released weather and GPS data to the public, it
fueled an industry that today is valued at tens of billions of dollars per year. Now,
weather and mapping tools are ubiquitous and help everyday Americans [navigate their
lives](business-case/).

The ultimate value of data can often not be predicted. That's why the U.S. Government
released a [policy](policy-memo/) that instructs agencies to manage their data, and
information more generally, as an asset from the start and, wherever possible,
release it to the public in a way that makes it open, discoverable, and usable.
```

Revision History



Changelog


PUBLIC
project-open-data / project-open-data.github.io
Watch 67
Unstar 220
Fork 114
branch: master
project-open-data.github.io / Commits
Sep 05, 2013
Merge pull request #129 from ultrasaurus/footnotes
haleyvandyck authored 12 days ago
e94a00d8fa
Browse code
Aug 25, 2013
No longer making "keyword" plural
MarinaMartin authored 22 days ago
4ddf05ae00
Browse code
Renamed person to ContactPoint in DRAFT changelog
MarinaMartin authored 23 days ago
87a32fc00d
Browse code
Aug 24, 2013
Merge pull request #126 from mhogeweg/patch-2
MarinaMartin authored 23 days ago
cd75882bad
Browse code
added nicer formatting to footnotes with link to anchor at bottom of ...
ultrasaurus authored 24 days ago
4f84dda90f
Browse code
fixed all bulleted lists, some links and spaces in content
rrbaker authored a month ago
ultrasaurus committed 24 days ago
3577ae40ed
Browse code
Aug 21, 2013
updated FGDC/ISO mapping to DCAT
mhogeweg authored a month ago
045c0a5478
Browse code

The Results

The Project's Pulse

March 17 2013 - September 17 2013

Period: **6 months**

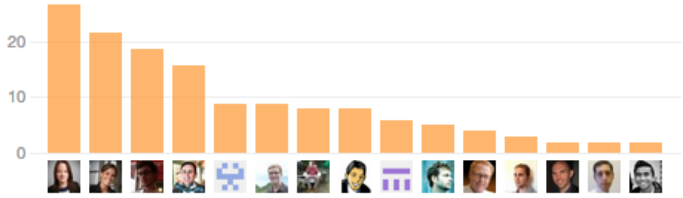
Overview

72 active pull requests

45 active issues

59 merged pull requests	**13** proposed pull requests	**33** closed issues	**12** new issues

52 authors have pushed **182 commits** to all branches, excluding merges. On master, **48 files** have changed and there have been **3,570 additions** and **1,030 deletions**.



Merged Pull Requests

59 Pull Requests merged by **34** people

merged #129 **linking to footnotes, with some formatting** 12 days ago

merged #126 **updated FGDC/ISO mapping to DCAT** 23 days ago

merged #122 **Added DataVizWiz Drupal module.** a month ago

merged #112 **Update governance.md** a month ago

merged #123 **fix the rendering of the unordered list under Short Description** a month ago

merged #124 **Fixing links to memo and schema** a month ago

merged #120 **Use kramdown as the markdown engine** a month ago

merged #119 **Updatedindex** a month ago

merged #118 **Implementation guidance** a month ago

merged #113 **Rename keyword field to "keywords"** a month ago

merged #114 **:fire: the size field** a month ago

merged #115 **Drop granularity** a month ago

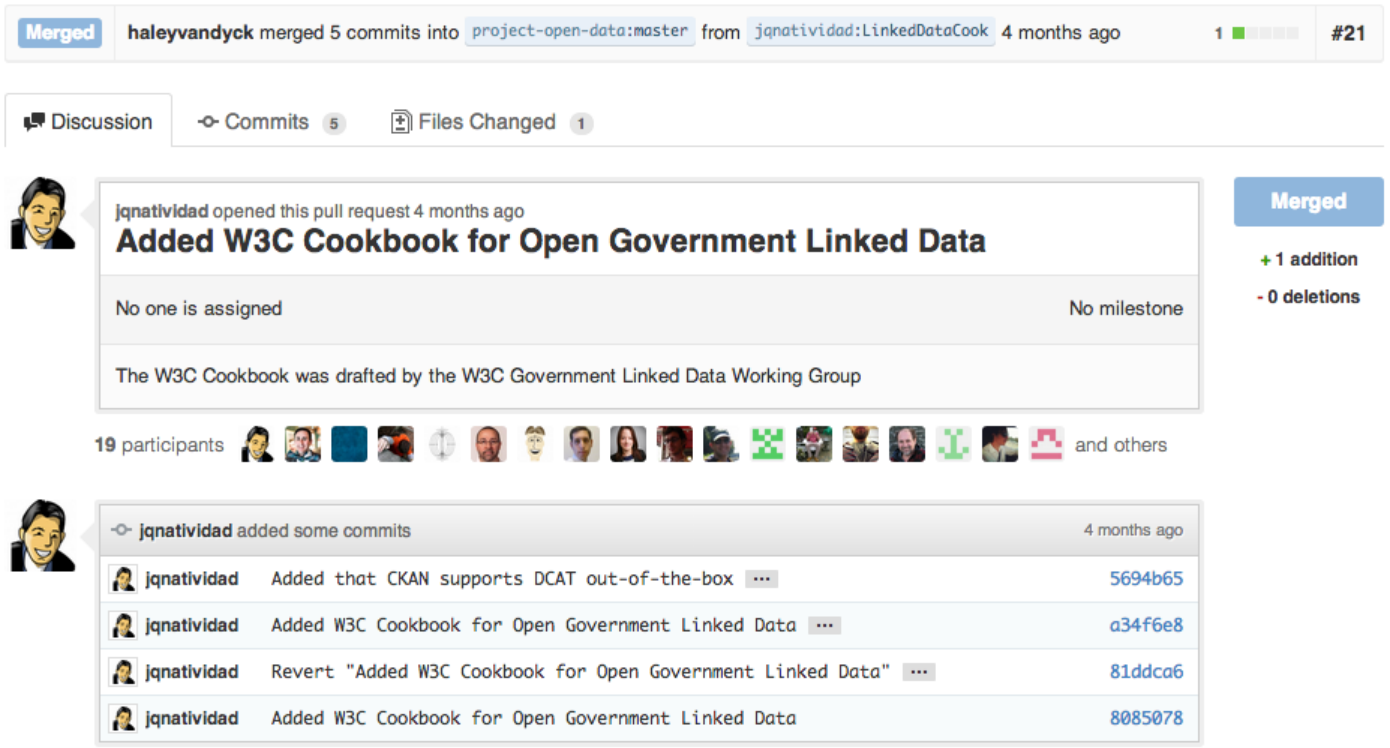
merged #80 **Updating the implementation guidance to include Socrata. Since, Socrata now offers built-in support for data.json** a month ago

merged #81 **Include a blurb for using Socrata.** a month ago

merged #87 **Use RDFa Lite 1.1 property attribute** a month ago

merged #46 **Change URLs to not include http://project-open-data.github.io (just use e.g. /schema/)** a month ago

An Example


Merged
haleyvandyck merged 5 commits into project-open-data:master from jqnatividad:LinkedDataCook 4 months ago
1
#21
Discussion
Commits 5
Files Changed 1
jqnatividad opened this pull request 4 months ago
Added W3C Cookbook for Open Government Linked Data
No one is assigned
No milestone
The W3C Cookbook was drafted by the W3C Government Linked Data Working Group
19 participants
and others
Merged
+ 1 addition
- 0 deletions
jqnatividad added some commits
4 months ago
jqnatividad Added that CKAN supports DCAT out-of-the-box ··· 5694b65
jqnatividad Added W3C Cookbook for Open Government Linked Data ··· a34f6e8
jqnatividad Revert "Added W3C Cookbook for Open Government Linked Data" ··· 81ddca6
jqnatividad Added W3C Cookbook for Open Government Linked Data 8085078

The Diff

Discussion | Commits 5 | Files Changed 1

Showing **1 changed file** with **1 addition** and **0 deletions**. **Show Diff Stats**

1 implementation-guide.md **View file @ 7969544**

```
...  ...  @@ -156,6 +156,7 @@ Collect or create information (data) in a way that supports downstream informati
156  156   * [NIST FIPS Publication 199](http://csrc.nist.gov/publications/fips/fips199/FIPS-PUB-199-final.pdf)
157  157   * [Controlled Unclassified Information requirements](http://www.archives.gov/cui/)
158  158   * [Mosaic Effect](http://www.computerworld.com/s/article/91109/Sidebar_The_Mosaic_Effect)
     159  ++ [W3C Cookbook for Open Government Linked Data](http://www.w3.org/2011/gld/wiki/Linked_Data_Cookbook)
159  160
160  161
161  162   ## 7) Release an Existing Data Collection as Open Data
```

The discussion

The Merge

georgethomas commented — 4 months ago

@benbalter 👍

haleyvandyck commented — 4 months ago

Thank you to everyone on this thread for the very interesting conversation. **@benbalter** said it best-- this is exactly why we are excited about Project Open Data, and the opportunity this provides for continual iteration and improvement.

Looks like there is agreement on adding **@jqnatividad** suggested link. 👍 Merging now.

Thank you all for contributing and we're looking forward to many more discussions and pull requests to come. 🇺🇸

- Haley, Senior Adviser at the White House

haleyvandyck referenced this pull request from a commit — 4 months ago

haleyvandyck `Merge pull request #21 from jqnatividad/LinkedDataCook` ··· ✔ `029a68e`

Merged **haleyvandyck** merged commit `029a68e` into `project-open-data:master` from `jqnatividad:LinkedDataCook` 4 months ago

Closed **haleyvandyck** closed the pull request 4 months ago