dim

drm inglorious maintainer script

Author:	Daniel Vetter <daniel.vetter@ffwll.ch>
Author:	Jani Nikula <jani.nikula@intel.com>
Date:	2014-05-15
Copyright:	2012-2016 Intel Corporation
Manual section:	1
Manual group:	maintainer tools

SYNOPSIS

dim [*option* ...] *command*

DESCRIPTION

dim is the Linux kernel drm subsystem maintainer script, used to maintain the drm-intel and drm-misc git repositories.

This man page is a command-line reference for the tool only; for more comprehensive documentation, including a getting started guide, see https://drm.pages.freedesktop.org/maintainer-tools/, or the same in source form at https://gitlab.freedesktop.org/drm/maintainer-tools.

OPTIONS

`-f`	Ignore some error conditions.
`-d`	Dry run.
`-i`	Interactive mode.

COMMANDS

The commands are grouped by target audiences and functionality below. Many of the commands have default aliases. See **dim list-aliases** for the list of aliases.

COMMANDS FOR DEVELOPERS

These commands can all be run without a full dim setup.

tc *commit-ish*

Print the oldest Linux kernel release or -rc tag that contains the supplied *commit-ish*, or, if none do, print the upstream branches that contain it.

cite *commit-ish*

Cite the supplied *commit-ish* in format 'sha1 ("commit subject")'.

fixes *commit-ish*

Print the Fixes: and Cc: lines for the supplied *commit-ish* in the linux kernel CodingStyle approved format.

checkpatch [*commit-ish* [.. *commit-ish*]] [*profile*]

Runs the given commit range commit-ish..commit-ish through the check tools.

If no commit-ish is passed, defaults to HEAD^..HEAD. If one commit-ish is passed instead of a range, the range commit-ish..HEAD is used.

If profile is given, uses specific options for checkpatch error filtering. Current profiles are "default", "branch", "drm-intel", and "drm-misc". The "branch" profile maps the current git branch to the appropriate profile, or if the branch is not known, to "default".

sparse [*commit-ish* [.. *commit-ish*]]

Run sparse on the files changed by the given commit range.

If no commit-ish is passed, defaults to HEAD^..HEAD. If one commit-ish is passed instead of a range, the range commit-ish..HEAD is used.

checker

Run sparse on drm/i915.

retip [*branch*] [*git-rebase option* ...]

Rebase the given local branch, current branch by default, onto drm-tip. Options after the branch will be passed to **git-rebase**.

range-diff [*commit-ish*] [*git-range-diff options*]

Convenience wrapper around the git range-diff command which automatically compares against HEAD if you only specify a commit-ish. In all other cases forwards to git range-diff. Defaults to @{1}, which is very useful for reviewing rebases. Additional options after the commit-ish will be passed to **git-range-diff**. Anything that can't be parsed as a commit-ish will also be forward in its entirety.

COMMANDS FOR COMMITTERS AND MAINTAINERS

setup *prefix*

Setup git maintainer branches in the given prefix.

update-branches

Updates all maintainer branches. Useful to synchronize all branches when other maintainers and committers pushed patches meanwhile.

cd

Changes the working directory into the git repository used by the last previous branch-specific command. This is implemented as a bash-function to make it useful in interactive shells and scripts. Only available when the bash completion is sourced.

checkout *branch*

Checks out the named branch.

cof

conf

conq

checkout shorthands for *drm-intel-fixes*, *drm-intel-next-fixes*, and *drm-intel-next-queued* branches respectively.

apply-branch branch [*git am arguments*]

Applys a patch to the given branch, complaining if it is not checked out yet.

apply-fixes [*git am arguments*]

apply-next-fixes [*git am arguments*]

apply-queued [*git am arguments*]

apply-branch shorthands for *drm-intel-fixes*, *drm-intel-next-fixes*, and *drm-intel-next-queued* branches respectively.

apply [*git am arguments*]

apply-branch shorthand for the current branch.

commit-add-tag *string* [...]

Append each argument at the end of the commit message of HEAD.

extract-tags *branch* [*git-rangeish*]

This extracts various tags (e.g. Reviewed-by:) from emails and applies them to the top commit on the given branch. You can give the command a rangeish to add the tags from the same email to multiple already applied patches.

extract-fixes [*git-rangeish*]

extract-next-fixes [*git-rangeish*]

extract-queued [*git-rangeish*]

extract-tags shorthands for *drm-intel-fixes*, *drm-intel-next-fixes*, and *drm-intel-next-queued* branches respectively.

push-branch branch [*git push arguments*]

Updates the named branch. Complains if that's not the current branch, assuming that patches got merged to the wrong branch. After pushing also updates linux-next and drm-tip branches.

push-fixes [*git push arguments*]

push-next-fixes [*git push arguments*]

push-queued [*git push arguments*]

push-branch shorthands for *drm-intel-fixes*, *drm-intel-next-fixes*, and *drm-intel-next-queued* branches respectively.

push [*git push arguments*]

push-branch shorthand for the current branch.

rebuild-tip

Rebuild and push the integration tree.

ADVANCED COMMANDS FOR COMMITTERS AND MAINTAINERS

cat-to-fixup [*branch*]

Pipes stdin into the fixup patch file for the current drm-tip merge. A branch can be explicitly specified to fix up a non-conflicting tree that fails to build.

magic-patch [-a]

Apply a patch using patch and then wiggle in any conflicts. When passing the option -a automatically changes the working directory into the git repository used by the last previous branch-specific command. This is useful with the per-branch workdir model.

add-link *branch*

This command adds the Link: tag (for patches that failed to apply directly).

add-link-fixes

add-link-next-fixes

add-link-queued

add-link shorthands for *drm-intel-fixes*, *drm-intel-next-fixes*, and *drm-intel-next-queued* branches respectively.

add-missing-cc

Adds all maintainers from scripts/get_maintainer.pl as cc's to the topmost commit. Any duplicates by name or email will be removed, so this can be used with *git rebase –exec "dim add-missing-cc"* to add cc's for an entire patch series that affect multiple drivers each with different maintainers.

magic-rebase-resolve

Tries to resolve a rebase conflict by first resetting the tree and then using the magic patch tool. Then builds the tree, adds any changes with git add -u and continues the rebase.

apply-resolved

Compile-test the current tree and if successful resolve a conflicted git am. Also runs the patch checker afterwards. This fails to add the Link: tag, so you'll need to add it manually or use the **add-link** subcommand.

create-branch *repo/branch* [*commit-ish*]

Create a new topic branch in the given *repo* named *branch*. The branch starts at HEAD or the given *commit-ish*. Note that topic/ is not automatically added to the branch name. Branch names should be unique across repos.

remove-branch *branch*

Remove the given topic branch.

create-workdir (*branch* | all)

Create a separate workdir for the branch with the given name, or for all branches if "all" is given.

for-each-workdir *command*

Run the given command in all active workdirs including the main linux kernel repository under $DIM_REPO.

update-driver-date *prefix file*

Update the the DRIVER_DATE and DRIVER_TIMESTAMP macros in *file* to match current date and time, and commit the change using given subject prefix.

COMMANDS FOR MAINTAINERS

cherry-pick *commit-ish* [*git cherry-pick arguments*]

Improved git cherry-pick version which also scans drm-tip for additional cherry-pick candidates. In dry-run mode/-d only the patch list is generated.

cherry-pick-fixes

cherry-pick-next-fixes

Look for non-upstreamed fixes (commits tagged Cc: stable@vger.kernel.org or Cc: drm-intel-fixes@lists.freedesktop.org) in drm-intel-next-queued, and try to cherry-pick them to drm-intel-fixes or drm-intel-next-fixes. These commands use dim cherry-pick internally to make sure bugfixes for fixes are cherry-picked too.

status

Lists all branches with unmerged patches, and how many patches are unmerged. It will show how the overall subsystem tree looks like and where patches waiting to be merged have been added, in order to help maintainers with deciding which tree is in need of a pull request. Commiters that want to check the status of their current branch should use normal **git status** commands.

pull-request *branch upstream*

Fetch the *upstream* remote to make sure it's up-to-date, create and push a date based tag for the *branch*, generate a pull request template with the specified *upstream*, and finally start $DIM_MUA with the template with subject and recipients already set.

Since the tag for the *branch* is date based, the pull request can be regenerated with the same commands if something goes wrong.

The tag will be signed using the key specified by $DIM_GPG_KEYID, if set.

pull-request-fixes [*upstream*]

pull-request shorthand for *drm-intel-fixes* as the branch and *origin/master* as the default upstream.

pull-request-next-fixes [*upstream*]

pull-request shorthand for *drm-intel-next-fixes* as the branch and *$DRM_UPSTREAM/drm-next* as the default upstream.

pull-request-next [*upstream*]

This is similar to **pull-request**, but for feature pull requests, with *drm-intel-next* as the branch and *$DRM_UPSTREAM/drm-next* as the default upstream.

The difference to **pull-request** is that this command does not generate a tag; this must have been done previously using **update-next**. This also means that the pull request can be regenerated with the same commands if something goes wrong.

apply-pull *branch*

Reads a pull request mail from stdin and merges it into the given *branch*.

backmerge *branch upstream*

Backmerges *upstream* into *branch*, making a few sanity checks on the way. The *upstream* we backmerge should be the same as used for sending out pull requests using **pull-request**. Alternatively it can also be a tag, which if available should be preferred.

rebase *branch upstream*

Rebases *branch* onto *upstream*, making a few sanity checks on the way. The *upstream* we rebase onto should be the same as used for sending out pull requests using **pull-request**. Alternatively it can also be a tag, which if available should be preferred.

update-next

Pushes out the latest dinq to drm-intel-next and tags it. For an overview a gitk view of the currently unmerged feature pile is opened.

The tag will be signed using the key specified by $DIM_GPG_KEYID, if set.

update-next-continue

When **update-next** fails to push the special release commit (because it raced with another committer) rebase and push manually, and then continue using this command.

tag-branch *branch* [*upstream*]

Pushes a new tag for the specified branch after checking that the remote is up-to-date.

The tag will be signed using the key specified by $DIM_GPG_KEYID, if set.

If upstream is provided, launch gitk to show the changes to be tagged.

tag-next

tag-branch shorthand for drm-intel-next.

Useful if drm-intel-next has been changed since the last run of the update-next command (e.g. to apply a hotfix before sending out the pull request).

DIM HELP COMMANDS

list-aliases

List all aliases for the subcommand names. Useful for autocompletion scripts.

See $dim_alias_<alias> under ENVIRONMENT below on how to define aliases.

list-branches

List all branches (main and topic) managed by dim. Useful for autocompletion scripts.

list-commands

List all subcommand names, including aliases. Useful for autocompletion scripts.

list-upstreams

List of all upstreams commonly used for pull requests. Useful for autocompletion scripts.

uptodate

Try to check if you're running an up-to-date version of **dim**.

help

Show this help. Install **rst2man(1)** for best results.

usage

Short form usage help listing all subcommands. Run by default or if an unknown subcommand was passed on the cmdline.

ALIASES

Extending dim functionalities

It is possible to create your own dim helper and aliases by adding them to $HOME/.dimrc:

```
dim_my_fancy_list_aliases()
{
        echo "Hello world!"
        dim_list_aliases
}

dim_alias_list_aliases=my-fancy-list-aliases
```

ENVIRONMENT

DIM_CONFIG

Path to the dim configuration file, $HOME/.dimrc by default, which is sourced if it exists. It can be used to set other environment variables to control dim.

DIM_PREFIX

Path prefix for kernel repositories.

DIM_REPO

The main linux kernel repository under $DIM_PREFIX.

DIM_MUA

Mail user agent. Must support the following subset of **mutt(1)** command line options: $DIM_MUA [-s subject] [-i file] [-c cc-addr] to-addr [...]

This is only needed for sending out pull requests.

DIM_MAKE_OPTIONS

Additional options to pass to **make(1)**. Defaults to "-j20".

DIM_TEMPLATE_HELLO

Path to a file containing a greeting template for pull request mails.

DIM_TEMPLATE_SIGNATURE

Path to a file containing a signature template for pull request mails.

DIM_GPG_KEYID

GPG key ID to use for signing tags. If set, tags will be signed. If unset, the default, tags will not be signed.

dim_alias_<alias>

Make **<alias>** an alias for the subcommand defined as the value. For example, *dim_alias_ub=update-branches*. There are some built-in aliases. Aliases can be listed using the **list-aliases** subcommand.

The alias functionality requires **bash(1)** version 4.3 or later to work.

EXAMPLES

Cross-subsystem topic branches

So you want to send a pull request to another subsystem? Maintainers will likely get cranky if you ask them to pull a swath of unrelated drm patches, so we'll use a topic branch based upon Linus' tree with only the relevant patches.

First select a suitable *baseline* for your topic branch. For topic branches shared within the gpu/drm subsystem, base it on the latest drm-next branch. For anything else, base it on the latest -rc tag from Upstream (not just any random position). In very rare cases you might need to apply topic branch pull requests from other maintainers before you can apply patches to construct a suitable baseline first.

Next, create the topic branch using dim. Use whichever dim remote is most applicable, and name the branch in a manner that describes the set of patches you want pulled. The upstream will be Linus' tree.

```
$ dim create-branch dim-remote/topic/topic-branch baseline
```

Once the branch is created, you can apply the patches to be pulled.

```
$ dim apply-branch topic/topic-branch
```

Test your new topic branch and push it.

```
$ dim push-branch topic/topic-branch
```

Ensure that your topic branch was merged into drm-tip. The drm-tip tree is located in $DIM_PREFIX/drm-tip, test it to ensure the new topic branch didn't break anything.

Once you're satisfied that nothing is broken, create the pull request.

```
$ dim pull-request topic/topic-branch baseline
```

You'll be prompted to enter a tag description and your mail user agent will open with the pull request email. Change names and emails as appropriate to reflect who the sender and recipient of the pull is, and send it.

Once the pull has been acked by your maintainer counterpart, you can pull it into the appropriate local dim branch.

```
$ dim apply-pull dim-branch
```

Perform a final test, and push *dim-branch* to *dim-remote*.

```
$ dim push-branch dim-branch
```

You can now remove the topic branch, as it is no longer useful (you could remove it any time after the pull request, since it creates a tag, but this is as good a place as any).

```
$ dim remove-branch topic/topic-branch
```