NAME

git-dpm - Debian packages in git manager

SYNOPSIS

git-dpm --help

git-dpm [*options*] *command* [*per-command-options and -arguments*]

DESCRIPTION

Git-dpm is a tool to handle a Debian source package in a git repository.

Each project contains three branches, a Debian branch (**master/***whatever*), a patched branch (**patched/patched-***whatever*) and an upstream branch (**upstream/upstream-***whatever*) and **git-dpm** helps you store the information in there so you have your changes exportable as quilt series.

Git-dpm will guess the other two branches based on the branch it sees. (Most commands act based on the current HEAD, i.e. what branch you have currently checked out, though some as e.g. **status** allows an optional argument instead). So for example, if you are in branch **master**, git-dpm assumes the corresponding upstream branch is called **upstream**. If you are in branch **upstream-something**, it assumes the Debian branch is called **something**.

Note that most commands may switch to another branch automatically, partly because it is easier to implement that way and hopefully so one does not need to switch branches manually so often.

SHORT EXPLANATION OF THE BRANCHES

the upstream branch (**upstream|upstream-***whatever*)

This branch contains the upstream sources. Its contents need to be equal enough to the contents in your upstream tarball.

the patched branch (**patched|patched-***whatever*)

This branch contains your patches to the upstream source. Every commit will be stored as a single patch in the resulting package.

Most of the time it will not exist as a branch known to **git**, but only as some point in the history of the Debian branch and possibly as tag for published versions. **Git-dpm** will create it when needed and remove the branch when no longer needed.

To help git generate a linear patch series, this should ideal be a linear chain of commits, whose description are helpful for other people.

As this branch is regularly rebased, you should not publish it.

the Debian branch (**master|***whatever***)**

This is the primary branch.

This branch contains the **debian/** directory and has the patched branch merged in.

Every change not in **debian/**, **.git*** or deleting files must be done in the patched branch.

EXAMPLES

Let's start with some examples:

Checking out a project

First get the master branch:
git clone *URL*

Then create upstream branch and see if the .orig.tar is ready:
git-dpm prepare

Create the patched branch and check it out:
git-dpm checkout-patched

Do some changes, apply some patches, commit them..
...
git commit

If your modification fixes a previous change (and that is not the last commit, otherwise you could have used --amend), you might want to squash those two commits into one, so use:
git rebase -i upstream

Then you want to get those changes into the Debian branch and the new patch files created (which you can do using **git-dpm update-patches**), but you most likely want to also document what you did in the changelog, so all in one step:
git-dpm dch -- -i

Perhaps change something in the Debian packaging:
...
git commit -a

Then push the whole thing back:
git push

Switching to a new upstream version

Get a new .orig.tar file. Either upgrade your upstream branch to the contents of that file and call **git-dpm record-new-upstream ../***new-stuff***.orig.tar.gz** or tell git-dpm to import and record it:
git-dpm import-new-upstream --rebase ../*new-stuff***.orig.tar.gz**

This will rebase the patched branch to the new upstream branch, perhaps you will need to resolve some conflicts:
vim ...
git add *resolved files*
git rebase --continue

After rebase is run (with some luck even in the first try):
git-dpm dch -- -v *newupstream***-1 "new upstream version"**

You could have also done the last step in three by:
git-dpm update-patches
dch -- -v *newupstream***-1 "new upstream version"**
git commit --amend -a

Do other debian/ changes:
...
git commit -a

Then push the whole thing back:

git push

Creating a new project

Create an **upstream** (or **upstream-***whatever*) branch containing the contents of your orig.tar file:
tar -xvf *example_0***.orig.tar.gz**
cd *example-0*
git init
git add .
git commit -m "import *example_0***.orig.tar.gz"**
git checkout -b upstream-unstable

You might want to use pristine tar to store your tar:
pristine-tar commit ../*example_0***.orig.tar.gz upstream-unstable**

Then let git-dpm know what tarball your upstream branch belongs to:
git-dpm init ../*example_0***.orig.tar.gz**

Note that since you were in **upstream-unstable** in this example, in the last example **git-dpm** assumed you want your Debian branch called **unstable** and not **master**, so after the command returned you are in the newly created **unstable** branch.

Do the rest of the packaging:
vim **debian/control debian/rules**
dch --create --package *example* **-v** *0***-1**
git add debian/control debian/rules debian/changelog
git commit -m "initial packaging"

Then add some patches:
git-dpm checkout-patched
vim ...
git commit -a
git-dpm dch "*fix ... (Closes: num)***"**

The **git-dpm checkout-patched** created a temporary branch **patched-unstable** (as you were in a branch called **unstable**. If you had called it with HEAD being a branch **master**, it would have been **patched**) to which you added commits. Then the **git-dpm update-patches** implied by **git-dpm dch** merged those changes into **unstable**, deleted the temporary branch and created new **debian/patches/** files.

Then build your package:
git-dpm status &&
dpkg-buildpackage -rfakeroot -us -uc -I".git*"

Now take a look what happened, perhaps you want to add some files to **.gitignore** (in the **unstable** branch), or remove some files from the **unstable** branch because your clean rule removes them.

Continue the last few steps until the package is finished. Then push your package:
git-dpm tag
git push --tags *target* **unstable:unstable pristine-tar:pristine-tar**

Removing existing patches

First get the master branch:
git clone *URL*

Create the patched branch and check it out:
git-dpm checkout-patched

Get a list of commits since the last upstream release: **git rebase -i upstream-unstable**

This will open your default editor with a list of commits. Edit the list to remove undesired commits.
...
git commit

Then you want to get those changes into the Debian branch and the old patch files deleted (which you can do using **git-dpm update-patches**), but you most likely want to also document what you did in the changelog, so all in one step:
git-dpm dch -- -i

Perhaps change something in the Debian packaging:
...
git commit -a

Then push the whole thing back:
git push

GLOBAL OPTIONS

--debug
Give verbose output what git-dpm is doing. Mostly only useful for debugging or when preparing an bug report.

--debug-git-calls
Output git invocations to stderr. (For more complicated debugging cases).

--allow-changes-in-debian-branch
Ignore upstream changes in your Debian branch. This will either discard them if merge-patched is called by come command or them being ignored elsewhere.

COMMANDS

init [*options*] *tarfile* [*upstream-commit* [*preapplied-commit* [*patched-commit*]]]
Create a new project.

The first argument is an upstream tarball.

You also need to have the contents of that file and the files given with --component unpackaged as some branch or commit in your git repository (or similar enough so **dpkg-source** will not know the difference). This will be stored in the upstream branch (called **upstream** or **upstream-***whatever*). If the second argument is non-existing or empty, that branch must already exist, otherwise that branch will be initialized with what that second argument. (It's your responsibility that the contents match. git-dpm does not know what your clean rule does, so cannot check (and does not even try to warn yet)).

You can already have an Debian branch (called **master** or **whatever**). If it does not exist, it will exist afterwards. Otherwise it can contain a **debian/patches/series** file, which git-dpm will import.

The third argument can be a descendant of your upstream branch, that contains the changes of

your Debian branch before any patches are applied (Most people prefer to have none and lintian warns, but if you have some, commit/cherry pick them in a new branch/detached head on top of your upstream branch and name them here). Without --patches-applied, your Debian branch may not have any upstream changes compared to this commit (or if it is not given, the upstream branch).

If there is no fourth argument, git-dpm will apply possible patches in your Debian branch on top of the third argument or upstream. You can also do so yourself and give that as fourth argument.

The contents of this commit/branch given in the fourth commit or created by applying patches on top of the third/your upstream branch is then merged into your Debian branch and remembered as patched branch.

Options:

--component *filename*
Record a **.orig-***component***.tar** file to be unpacked in your upstream branch.

--patches-applied
Denote the Debian branch already has the patches applied.

Without this git-dpm will check there are no changes in the Debian branch outside patch management before applying the patches; with this, it will instead check there are no differences after applying the patches.

--create-no-patches
Do not create/override **debian/patches** directory. You will have to call **update-patches** yourself. Useful if you are importing historical data and keep the original patches in the Debian branch.

--record-patch-category
Add a **Patch-Category:** field to each imported patch that is in a subdirectory of **debian/patches**. This causes **update-patches** to store it in the same subdirectory.

--record-patch-name
Add a **Patch-Name:** field to each imported patch with its name. This causes **update-patches** to store it under its original name.

prepare
Make sure upstream branch and upstream orig.tar ball are there and up to date. (Best called after a clone or a pull).

status [*branch*]

Check the status of the current project (or of the project belonging to the argument *branch* if that is given). Returns with non-zero exit code if something to do is detected.

checkout-patched

Checkout the patched branch (**patched**|**patched-***whatever*) after making sure it exists and is one recorded in the **debian/.git-dpm** file.

If the patched branch references an old state (i.e. one that is already ancestor of the current Debian

branch), it is changed to the recorded current one.

Otherwise you can reset it to the last recorded state with the **--force** option.

update-patches [*options*] [*branch-name*]

After calling **merge-patched-into-debian** if necessary, update the contents of **debian/patches** to the current state of the **patched** branch.

Also record in debian/.git-dpm which state of the patched branch the patches directory belongs to.

If a *branch-name* is given, that branch is processed. Otherwise the name is derived from the currently checked out branch as usual.

Options:

--redo Do something, even if it seems like there is nothing to do.

--allow-revert, **--ignore-deletions**, **--dot-git-files=***
passed on to merge-patched-into-debian

--amend
Do not create a new commit, but amend the last one in the Debian branch. (I.e. call merge-patched-into-debian with --amend and amend the updates patches into the last commit even if that was not created by merge-patched-into-debian).

-m *message*
Use *message* as commit message. (If used together with --amend, do not reuse old commit message, author or author date but replace the old commit with a new commit with that message).

--keep-branch
do not remove an existing patched branch (usually that is removed and can be recreated with **checkout-patched** to avoid stale copies lurking around.

--allow-nonlinear
passed to merge-patched.

dch [*options*] **--** *dch-options*
After calling update-patches if necessary, run devscripts' dch with the specified options and then do a **git commit** with a commit message containing changes to the **debian/changelog** file.

Options:

--amend
Replace the commit currently the head of the Debian branch (**master**|*something*) instead of creating a new one on top. The commit message will also include changes done to **debian/changelog** in the previous commit (unless reverted by the new edit).

--ignore-patches
Do not call update-patches but simply ignore the current state of the patched branch (**patched**|**patched-***something*).

--keep-branch, **--allow-revert**, **--allow-nonlinear**, **--ignore-deletions**, **--dot-git-files=***
Passed to update-patches, if called.

--latest-only|--latest|-l
Only include changes between the current working directory before calling dch and after calling it (and not since the last commit or the last commit not replaced).

-e | -v | -a | --all | -s | -n | --no-verify | -u | --untracked-files | -q | --quiet | --cleanup=... | **--author=**...
passed to git commit.

merge-patched-into-debian [*options*] [*branch-name*]
Usually **update-patches** runs this for you if deemed necessary.

This command is the core of **git-dpm**, but you usually do not call it directly. It is called by **update-patches** and things calling **update-patches** like **dch** when necessary.

It replaces all files (with only the exceptions described below) in the current Debian branch (**master|***whatever*) with those found in the patched branch (**patched|patched-***whatever*).

Only the **debian** directory and files in the root directory starting with ".git" are kept from the Debian branch (so **.gitignore**, **.gitattributes**, ... will stay). And all files that were found in the last recorded patched branch and deleted in the current Debian branch will also be deleted in the new one.

Additionally the **debian/.git-dpm** file will be updated so the current patched branch is recorded and is marked as belonging to the last recorded upstream branch.

If there is no *branch-name* given on the command line the base name of the branches to operate on is computed from the currently checked out branch as usual. Otherwise this argument is used.

Options:

--allow-revert
Usually reverting to an old state of the patched branch is not allowed, to avoid mistakes (like having only pulled the Debian branch and forgot to run **checkout-patched**). This option changes that so you can for example drop the last patch in your stack.

--no-ignore-deletions (default)
Files deleted currently in the Debian branch relative to the recorded patched branch will still be deleted in the new Debian branch and not taken from the new patched branch. This is the default unless a different default was set with
git config dpm.*BRANCHNAME***.dpmIgnoreDeletions true**.

--ignore-deletions
Disable the behavior described in **--no-ignore-deletions**.

--dot-git-files=*method*
Specify how files starting with **.git** outside **debian/** are handled. Those are handles specially as **.gitattributes** and **.gitignore** might be different in the Debian branch without being part of any patch. (The whole **debian/** directory is always taken from the Debian branch, so files there are not affected).

Possible methods are:

automatic (default)
Any **.git*** files that are added, modified or removed in the current Debian branch compared to the old upstream branch are set to this state, everything else is taken as found in the new patched branch.

debian All **.git*** files are taken from the Debian branch. Files with a name like that from the patched branch are ignored.

upstream
Files starting with **.git** are not given special handling. They are taken from the patched branch, unless they are deleted in the Debian branch and the default **--no-ignore-deletions** is active. (i.e. just like any other file outside **debian/**).

--keep-branch
do not remove an existing patched branch (usually that is removed and can be recreated with **checkout-patched** to avoid stale copies lurking around).

--amend
Replace the last commit on your Debian branch (as git commit --amend would do). With the exception that every parent that is an ancestor of or equal to the new patched branch or the recorded patched branch is omitted. (That is, you lose not only the commit on the Debian branch, but also a previous state of the patched branch if your last commit also merged the patched branch).

-m *message*
Commit message to use for the new commit created. (If used together with --amend, this disables reusing the old author and date).

--allow-nonlinear
do not abort with an error if the patched branch is no linear series of commits on top of the upstream branch. Using this option is not recommended as it easily hides problems with patched or upstream branch and may introduce broken debian/patches/ series, as format-patch does no serialisation.

import-new-upstream [*options*] *.orig.tar*
Import the contents of the given tarfile (as with **import-tar**) and record this branch (as with **record-new-upstream**).

This is roughly equivalent to:
git-dpm import-tar -p *upstream filename*
git checkout -b *upstream*
git-dpm record-new-upstream *filename*

--detached
Don't make the new upstream branch an ancestor of the old upstream branch (unless you re-add that with **-p**).

-p *commit-id*|**--parent** *commit-id*
Give **import-tar** additional parents of the new commit to create.

For example if you track upstream's git repository in some branch, you can name that here to make it part of the history of your Debian branch.

--allow-no-parent
If dpm.importWithoutParent is set to false via git config, git-dpm will not allow import-new-upstream to be run without this option or at least on -p option.

--rebase-patched
After recording the new upstream branch, rebase the patched branch to the new upstream branch.

--no-rebase-patched
Do not call rebase-patched after recording the new upstream branch. (This is currently the default, but that may change in the future).

-m *message*
Commit message to use for the new commit to the Debian branch recording the new file and upstream branch.

--component *package_version*.**orig-**.*component*.**tar**.*gz*
Unpack the specified filename into the *component* directory and record it so that **prepare** and **status** know to check for it.

--init

None of the branches yet exists, create them.

As the branches to operate on are derived from **HEAD** if no **--branch** option is given, you either need **HEAD** point to an not yet existing branch (like directly after **git init**) or you need you give a name with **--branch**. Otherwise one of the branches already exists and you only get an error message.

--branch *debianbranch*
Don't derive the Debian branch name from current **HEAD** but use *debianbranch* instead. (And upstream branch name and patched branch name derived from that as usual).

--pristine-tar-commit | --ptc
Call **pristine-tar commit** for all imported tarballs not yet found in the pristine-tar branch.

--no-pristine-tar-commit
Do not call **pristine-tar commit** for all imported tarballs even if configured to do so by
git config dpm.pristineTarCommit true or by
git config branch.debianbranch**.dpmPristineTarCommit true**.

--ignore-deletions, **--dot-git-files=**
Passed to merge-patched, if called (only done if there were no patches previously).

--upstream-author *author*
Used as the **--author** argument to **git-dpm import-tar**.

--upstream-date *date*
Used as the **--date** argument to **git-dpm import-tar** (especially **auto** is supported to extract a date from the tar file).

--exclude *pattern*
The given pattern is passed to tar as exclude pattern when unpacking. Can be given multiple times.

import-tar [*options*] *.tar-file*
Create a new commit containing the contents of the given file. The commit will not have any parents, unless you give **-p** options.

-p *commit-id*|**--parent** *commit-id*
Add the given commit as parent. (Can be specified multiple times).

--branch *branchname*
Create new branch *branchname* if it does not already exist or replace *branchname* with a commit created from the tarball with the current *branchname* head as parent.

-m *message*
Do not start an editor for the commit message, but use the argument instead.

--date *date*
Date of the commit to create.

If the value is **auto** then the newest date of any file or directory in the tarball is used.

--author *author*
Author of the commit to create. It has to be in the usual git format
author **<***email***>**.

--exclude *pattern*
The given pattern is passed to tar as exclude pattern when unpacking. Can be given multiple times.

record-new-upstream [*options*] *.orig.tar* [*commit*]

If you changed the upstream branch (**upstream**|**upstream-***whatever*), git-dpm needs to know which tarball this branch now corresponds to and you have to rebase your patched branch (**patched**|**patched-***whatever*) to the new upstream branch.

If there is a second argument, this command first replaces your upstream branch with the specified commit.

Then the new upstream branch is recorded in your Debian branch's **debian/.git-dpm** file.

If you specified **--rebase-patched** (or short **--rebase**), **git-dpm rebase-patched** will be called to rebase your patched branch on top of the new upstream branch.

After this (and if the branch then looks like what you want), you still need to call **git-dpm merge-patched-into-debian** (or directly **git-dpm update-patches**).

WARNING to avoid any misunderstandings: You have to change the upstream branch before using this command. It's your responsibility to ensure the contents of the tarball match those of the upstream branch.

--rebase-patched
Automatically call **git-dpm rebase-patched**.

--new-tarball-only
Don't refuse operation if the tarball changes but the upstream branch did not. (This is only sensible if the tarball changed without changing its contents, see the warning above).

-m *message*
Commit message to use for the new commit to the Debian branch recording the new file and upstream branch.

--amend
Replace the last commit instead of creating a new one on top.

--component *filename*
Record *filename* as needed component source file (i.e. a *sourcename_upstreamversion*.**orig-***component*.**tar.***compression* file). It's your responsible to have that file's contents already as part of your upstream branch (in a *component* subdirectory).

(Recorded files will be looked for by **status** and **prepare**. The list of recorded component source files is removed when a new upstream branch or upstream **.orig** source file is recorded).

--ignore-deletions, **--ot-git-files=**
Passed to merge-patched, if called (which is only done if there were no patches previously, so the new upstream branch is merged in directly).

rebase-patched
Try to rebase your current patched branch (**patched**|**patched-***whatever*) to your current current upstream branch (**upstream**|**upstream-***whatever*).

If those branches do not yet exist as git branches, they are (re)created from the information recorded in **debian/.git-dpm** first.

This is only a convenience wrapper around git rebase that first tries to determine what exactly is to rebase. If there are any conflicts, git rebase will ask you to resolve them and tell rebase to continue.

After this is finished (and if the branch then looks like what you want), you still need **merge-patched-into-debian** (or directly **update-patches**).

tag [*options*] [*version*]
Add tags to the upstream, patched and Debian branches. If no version is given, it is taken from debian/changelog.

Options:

--refresh
Overwrite the tags if they are already there and differ (except upstream).

--refresh-upstream
Overwrite the upstream if that is there and differs.

--allow-stale-patches
Don't error out if patches are not up to date. This is only useful if you are importing historical data and want to tag it.

--named
Use the package name as part of the names of the generated tags. (use **git config dpm.tagsNamed true** to make this the default)

--with-name *name*
Like **--named** but give the name to use.

--debian-tag *tag-name*

--patched-tag *tag-name*

--upstream-tag *tag-name*
Specify the names of the tags to generate.

%p is replaced with the package name,

%v with the version (without epoch) with colons (:) and tilde (~) replaced by underscore (_),

%u with the upstream version (without epoch or Debian revision) with colons (:) and tilde (~) replaced by underscore (_),

%e with the epoch,

%f with the epoch followed by an underscore (_) if there is an epoch, and with the empty string if there is no epoch,

%V with the version (without epoch) with colons (:) and tilde (~) replaced by dots (.),

%U with the upstream version (without epoch or Debian revision) with colons (:) and tilde (~) replaced with dots (.),

%E with the epoch followed by a dot if there is an epoch, and with the empty string if there is no epoch,

%% with a single **%**.

If one of those is not set via the command line option, **git config** is asked about value of **dpm.debianTag**, **dpm.patchedTag** or **dpm.upstreamTag**. If that is also not set or the special value **AUTO**, then debian/.git-dpm is scanned for a line of the form
debianTag="*value***"**,
patchedTag="*value***"** or
upstreamTag="*value***"**.

(Note: always add those to the end of the file, the first eight lines have fixed line numbers)

If this still does not result in an pattern to use, the defaults are '**%p-debian%e-%v**', '**%p-patched%e-%v**' and '**%p-upstream%e-%u**' with **--named** and '**debian%e-%v**',

'**patched%e-%v**' and '**upstream%e-%u**' without.

If a tag name has the special value **NONE**, no tag is generated.

ref-tag [*options*] *commit* [*version*]
Like **tag**, but create tags for *commit*, i.e. *commit* will get the Debian tag and the other tags are placed where the **debian/.git-dpm** file of that commit points to.

So it is mostly equivalent to:
git checkout -b temp *commit*
git-dpm tag [*options*] [*version*]
git checkout *previous-head*
git branch -D temp

Options like **tag**.

apply-patch [*options...*] [*filename*]
Switch to the patched branch (assuming it is up to date, use checkout-patched first to make sure or get an warning), and apply the patch given as argument or from stdin.

--author *author <email>*
Override the author to be recorded.

--defaultauthor *author <email>*
If no author could be determined from the commit, use this.

--date *date*
Date to record this patch originally be from if non found.

--dpatch
Parse patch as dpatch patch (Only works for dpatch patches actually being a patch, might silently fail for others).

--cdbs Parse patch as cdbs simple-patchsys.mk patch (Only works for dpatch patches actually being a patch, might silently fail for others).

--edit Start an editor before doing the commit (In case you are too lazy to amend).

--record-name
Add a **Patch-Name:** field to tell **update-patches** to export it with the same name again.

--name *name*
Add a **Patch-Name:** field to tell **update-patches** to use *name* as filename to store this patch into (relative to **debian/patches**).

--category *name*
Add a **Patch-Category:** field to tell **update-patches** to always export this patch into a subdirectory *name* of **debian/patches**.

cherry-pick [*options...*] *commit*
Recreate the patched branch and cherry-pick the given commit. Then merge that back into the Debian branch and update the debian/patches directory (i.e. mostly equivalent to checkout-patched, git's cherry-pick, and update-patches).

--merge-only
Only merge the patched branch back into the Debian branch but do not update the patches directory (You'll need to run update-patches later to get this done).

-e | --edit
Passed to git's cherry-pick: edit the commit message picked.

-s | --signoff
Passed to git's cherry-pick: add a Signed-off-by header

-x Passed to git's cherry-pick: add a line describing what was picked

-m *num* | **--mainline** *num*
Passed to git's cherry-pick: allow picking a merge by specifying the parent to look at.

--repick
Don't abort if the specified commit is already contained.

--allow-nonlinear, **--ignore-deletions**, **--dot-git-files=**
Passed to update-patches, if called.

passed to merge-patched-into-debian and update-patches.

--keep-branch
do not remove the patched branch when it is no longer needed.

--amend
passed to merge-patched-into-debian: amend the last commit in the Debian branch.

import-dsc
Import a Debian source package from a .dsc file. This can be used to create a new project or to import a source package into an existing project.

While a possible old state of a project is recorded as parent commit, the state of the old Debian branch is not taken into account. Especially all file deletions and .gitignore files and the like need to be reapplied/re-added afterwards. (Assumption is that new source package versions from outside might change stuff significantly, so old information might more likely be outdated. And reapplying it is easier then reverting such changes.)

First step is importing the **.orig.tar** file and possible **.orig-***component***.tar** files. You can either specify a branch to use. Otherwise **import-dsc** will look if the previous state of this project already has the needed file so the old upstream branch can be reused. If there is non, the file will be imported as a new commit, by default with a possible previous upstream branch as parent.

Then **import-dsc** will try to import the source package in the state as **dpkg-source -x** would create

it. (That is applying the .diff and making **debian/rules** executable for 1.0 format packages and replacing the **debian** directory with the contents of a .debian.tar and applying possible **debian/patches/series** for 3.0 format packages). This is later referred to as verbatim import.

If it is a 1.0 source format package, **import-dsc** then looks for a set of supported patch systems and tries to apply those patches. Those are then merged with the verbatim state into the new Debian branch.

Then a **debian/.git-dpm** file is created and a possible old state of the project added as parent.

Note that **dpkg-source** is not used to extract packages, but they are extracted manually. Especially **git-apply** is used instead of **patch**. While this generally works (and **git-dpm** has some magic to work around some of **git-apply**'s shortcomings), unclean patches might sometimes need a **-C0** option and then in same cases be applied at different positions than where **patch** would apply them.

General options:

-b | --branch *branch-name*
: Don't look at the current HEAD, but import the package into the git-dpm project *branch-name* or create a new project (if that branch does not yet exist).

--verbatim *branch-name*
: After **import-dsc** has completed successfully, *branch-name* will contain the verbatim import of the .dsc file. If a branch of that name already exists, the new verbatim commit will also have the old as parent. (This also causes the verbatim commit not being amended with other changes, which can result in more commits).

--use-changelog
: Parse debian/changelog of the imported package. Use the description as commit messages and the author and time as default for patches and import commits without that information. (Warning: may still contain some rough edges).

Options about creating the upstream branch:

--upstream-to-use *commit*
: Do not import the .orig.tar nor try to reuse an old import, but always use the *commit* specified.

  It is your responsibility that this branch is similar enough to the .orig.tar file plus possible .orig-component.tar in their respective directories. (As usual, similar enough means: Does not miss any files that your patches touch or your build process requires (or recreates unless **debian/rules clean** removes them again). Every file different than in .orig.tar or not existing there you must delete in the resulting Debian branch. No patch may touch those files.)

  Use with care. Nothing will warn you even if you use the contents of a totally wrong upstream version.

--detached-upstream
: If importing a .orig.tar as new commit, do not make an possible commit for an old upstream version parent.

--upstream-parent *commit*
Add *commit* as (additional) parent if importing a new upstream version.

(This can for example be used to make upstream's git history part of your package's history and thus help git when cherry-picking stuff).

--allow-no-parent
If dpm.importWithoutParent is set to false via git config, git-dpm will not allow import-dsc to be run without this option or at least on --upstream-parent option.

--pristine-tar-commit |--ptc
Call **pristine-tar commit** for all tarballs imported after the rest of the import-dsc command was successful.

--no-pristine-tar-commit
Do not call **pristine-tar commit** for all imported tarballs even if configured to do so by
git config dpm.pristineTarCommit true or by
git config branch.debianbranch**.dpmPristineTarCommit true**.

--upstream-author *author*
Used as the **--author** argument to **git-dpm import-tar**.

--upstream-date *date*
Used as the **--date** argument to **git-dpm import-tar** (especially **auto** is supported to extract a date from the tar file).

--tar-exclude *pattern*
The given pattern is passed to tar as exclude pattern when unpacking tarfiles. Can be given multiple times.

Options about applying patches:

-f | --force-commit-reuse
Only look at parent and tree and no longer at the description when trying to reuse commits importing patches from previous package versions.

-C*num* **| --patch-context** *num*
Passed as **-C***num* to **git-apply**. Specifies the number of context lines that must match.

--dpatch-allow-empty
Do not error out if a dpatch file does not change anything when treated as patch.

As dpatch files can be arbitrary scripts, **git-dpm** has some problems detecting if they are really patches. (It can only cope with patches). If a script that is not a patch is treated as patch that usually results in patch not modify anything, thus those are forbidden without this option.

--patch-system *mode*
Specify what patch system is used for source format 1.0 packages.

auto (this is the default)
Try to determine what patch system is used by looking at **debian/rules** (and **debian/control**).

none Those are not the patches you are looking for.

history Don't try to find any patches in the .diff (like **none**). If if the project already exists and the upstream tarball is the same, create the patched state of the new one by using the patches of the old one and adding a patch of top bringing it to the new state.

If you import multiple revisions of some package, where each new revision added at most a single change to upstream, this option allows you to almost automatically create a proper set of patches (ideally only missing descriptions).

If there are same changes and reverts those will be visible in the patches created, so this mode is not very useful in that case.

quilt Extract and apply a **debian/patches/series** quilt like series on top of possible upstream changes found in the .diff file.

quilt-first
As the **quilt** mode, but apply the patches to an unmodified upstream first and then cherry-pick the changes found in the .diff file.

As this is not the order in which patches are applied in a normal unpack/build cycle, this will fail if those changes are not distinct enough (for example when patches depend on changes done in the .diff).

But if the .diff only contains unrelated changes which varies with each version, this gives a much nicer history, as the commits for the patches can more easily be reused.

quilt-applied
As the **quilt-first** mode, but assume the patches are already applied in the .diff, so apply them on top of an unmodified upstream and then add a commit bringing it to the state in the .diff. (Or not if that patch would be empty).

dpatch | dpatch-first | dpatch-applied
Like the **quilt** resp. **quilt-first** resp. **quilt-applied** modes, but instead look for dpatch-style patches in **debian/patches/00list**.

Note that only patches are supported and not dpatch running other commands.

simple | simple-first | simple-applied
Like the **quilt** resp. **quilt-first** resp. **quilt-applied** modes, but instead assume **debian/patches/** contains patches suitable for cdbs's **simple-patchsys.mk**.

--patch-author "*name* <*email*>"
Set the author for all git commits importing patches.

--patch-default-author "*name* <*email*>"
Set an author for all patches not containing author information (or where **git-dpm** cannot determine it).

--edit-patches
For every patch imported, start an editor for the commit message.

--record-patch-category
Add a **Patch-Category:** field to each imported patch that is in a subdirectory of **debian/patches**. This causes **update-patches** to store it in the same subdirectory.

--record-patch-name
Add a **Patch-Name:** field to each imported patch with its name. This causes **update-patches** to store it under its original name.

record-dsc [*options*] *commit .dsc-file*
Store a pristine .dsc file in a **dscs** branch after storing the files it contains using pristine-tar.

The first argument is an tag or commit storing the **git-dpm** project in the state belonging to the **.dsc** file and the second argument is the **.dsc** file itself. The files it references are expected in the same directory as the file itself (if they are needed).

Some checks are done to make sure the file and its contents are named properly and match the commit in question, but only cursory to avoid obvious mistakes (for example only the version is checked, but .debian.tar is not unpacked to check the files are really the same, for example).

Options:

--create-branch
Create a new **dscs** branch.

--allow-unsigned
Allow recording a unsigned **.dsc** file. This usually defeats the point of storing them at all.

the debian/.git-dpm file

You should not need to know about the contents if this file except for debugging git-dpm.

The file contains 8 lines, but future version may contain more.

The first line is hint what this file is about and ignored.

Then there are 4 git commit ids for the recorded states:

First the state of the patched branch when the patches in **debian/patches** were last updated.

Then the state of the patched branch when it was last merged into the Debian branch.

Then the state upstream branch when the patched branch was last merged.

Finally the upstream branch.

The following 3 lines are the filename, the sha1 checksum and the size of the origtarball belonging to the recorded upstream branch.

SHORTCUTS

Most commands also have shorter aliases, to avoid typing:

update-patches: up, u-p, ci
prepare: prep

checkout-patched: co, c-p
rebase-patched: r-p
apply-patch: a-p
import-tar: i-t
import-new-upstream: i-n-u, inu
record-new-upstream: r-n-u, rnu
merge-patched-in-debian: merge-patched

record-new-upstream is also available under the old name new-upstream, though likely will be removed in future versions (to avoid confusion).

BRANCHES

the upstream branch (**upstream|upstream-***whatever*)

This branch contains the upstream sources. It contents need to be equal enough to the contents in your upstream tarball.

Equal enough means that dpkg-source should see no difference between your patched tree and and original tarball unpackaged, the patched applied and **debian/rules clean** run. Usually it is easiest to just store the verbatim contents of your orig tarball here. Then you can also use it for pristine tar.

This branch may contain a debian/ subdirectory, which will usually be just ignored.

You can either publish that branch or make it only implicitly visible via the **debian/.git-dpm** file in the Debian branch.

While it usually makes sense that newer upstream branches contain older ones, this is not needed. You should be able to switch from one created yourself or by some foreign-vcs importing tool generated one to an native upstream branch or vice versa without problems. Note that since the Debian branch has the patched branch as ancestor and the patched branch the upstream branch, your upstream branches are part of the history of your Debian branch. Which has the advantage that you can recreate the exact state of your branches from your history directly (like **git checkout -b oldstate** *myoldtagorshaofdebianbranchcommit* **; git-dpm prepare ; git checkout unstable-oldstate**) but the disadvantage that to remove those histories from your repository you have to do some manual work.

the patched branch (**patched|patched-***whatever*)

This branch contains your patches to the upstream source. (which of course means it is based on your upstream branch).

Every commit will be stored as a single patch in the resulting package.

To help git generate a linear patch series, this should ideal be a linear chain of commits, whose description are helpful for other people.

As this branch is regularly rebase d, you should not publish it. Instead you can recreate this branch using **git-dpm checkout-patched** using the information stored in **debian/.git-dpm**.

You are not allowed to change the contents of the **debian/** subdirectory in this branch. Renaming files or deleting files usually causes unnecessary large patches.

the Debian branch (**master**|*whatever*)
: This is the primary branch.

  This branch contains the **debian/** directory and has the patched branch merged in.

  Every change not in **debian/**, **.git*** or deleting files must be done in the patched branch.

alternative branch names
: You can specify alternate branch names for upstream and patched branches of a specific Debian branch, or force a branch to be a Debian branch that would normally be considered e.g. upstream branch of another branch by adding **dpmUpstreamBranch** and **dpmPatchedBranch** configure items for the Debian branch in question (you need both, only one is treated as error).

  The following example is a no-op for all practical purposes:
  git config branch.master.dpmUpstreamBranch upstream
  git config branch.master.dpmPatchedBranch patched

COPYRIGHT