

**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 a 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 ideally be a linear chain of commits, whose descriptions 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
```

Merge your changes into the debian branch and create patches:

```
git-dpm update-patches
```

```
dch -i
```

```
git commit --amend -a
```

Perhaps change something with the debian package:

...

```
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 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 update-patches
```

Record it in debian/changelog:

```
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 update-patches
dch "fix ... (Closes: num)"
git commit --amend -a
```

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** merged that 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*"
```

Not 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
```

## 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).

**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 those (or similar enough so **dpkg-source** will not know the difference) as some branch or commit in your git repository. 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 forth 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 forth argument.

The contents of this commit/branch given in the forth 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:

**--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 but 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.

**--no-commit**

Do not commit the new **debian/.git-dpm** file and possible **debian/patched** changes, but only add them to working tree and index.

**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**

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.

Options:

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

**--allow-revert**  
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**).

**--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).

**merge-patched-into-debian**

Usually **update-patches** runs this for you if deemed necessary.

Replace the current contents of the debian branch (**master|*whatever***) with the contents of the patched branch (**patched|patched-*whatever***), except for everything under **debian/**. Also files that are deleted in the debian branch keep being deleted and files in the root directory starting with ".git" keep their contents from the debian branch, too.

The current state of the patched branch is recorded in **debian/.git-dpm** and so is which upstream branch was recorded patched branch is relative to (to easy future **merge-patched-into-debian**

operations).

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.

**--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).

**import-new-upstream** [*options*] *.orig.tar*

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

This is roughly equivalent to:

```
git-dpm import-tar -p upstream filename
git checkout -b upstream
git-dpm new-upstream filename
```

**--detached**

Don't make the new upstream branch an ancestor of the old upstream branch (unless you readd 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.

**--rebase-patched**

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

**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).

**-m** *message*

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

**new-upstream** [**--rebase-patched**] *.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**

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** [*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-nonclean**

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.

**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).

**--cdb** Parse patch as `cdb 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).

**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 specifign the parent to look at.

**--repick**

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

**--allow-nonlinear**

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/readded 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. 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` executeable 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 some 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).

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 in question. (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).

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.

**-Cnum** | **--patch-context** *num*

Passed as **-Cnum** 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 cdb'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.

### **the debian/git-dpm file**

You should not need to know about the contents of 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
new-upstream-branch: new-upstream, n-u
apply-patch: a-p
import-tar: i-t
import-new-upstream: i-n-u, inu
cherry-pick: c-p
```

## 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.

Equal enough means that `dpkg-source` should see no difference between your patched tree 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 a 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 rebased, 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 usuall causes unecessary 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.

## **COPYRIGHT**

Copyright © 2009,2010 Bernhard R. Link

This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

## **REPORTING BUGS AND ISSUES**

You can report bugs or feature suggestions to [gi-dpm-devel@lists.alioth.debian.org](mailto:gi-dpm-devel@lists.alioth.debian.org) or tome. Please send questions to [git-dpm-user@lists.alioth.debian.org](mailto:git-dpm-user@lists.alioth.debian.org) or to me at [brlink@debian.org](mailto:brlink@debian.org).