## 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−***whaterver*) 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.

## SHORT EXPLANATION OF THE 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.

the patched branch (**patched**|**patched−***whaterver*)

This branch contains your patches to the upstream source. 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 regulary rebased, you should not publish it.

the debian branch (**master**|*whaterver***)**

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/changes:
**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**

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
```

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 becaus 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.

## 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 responsiblity 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**

Check the status of the current project. Returns with non-zero exit code if something to do is detected.

**checkout−patched**

Checkout the patched branch (**patched**|**patched−***whaterver*) 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**
passed on to 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**|*whaterver*) with the contents of the patched branch (**patched|patched−***whaterver*), 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).

**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−***whaterver*) 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−***whaterver*) 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 resolv 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 uptream, 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.

**−−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.

## the debian/.git−dpm file

You should not need to know about the contents if this file except for debuging 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.

## 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−***whaterver*)

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 regulary 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 unecesary large patches.

the debian branch (**master**|*whaterver*)

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

## REPORTING BUGS AND ISSUES

You can report bugs or feature suggestions to git-dpm-devel@lists.alioth.debian.org or tome. Please send questions to git-dpm-user@lists.alioth.debian.org or to me at brlink@debian.org.