NAME

darcs − an advanced revision control system

SYNOPSIS

darcs *COMMAND*...

DESCRIPTION

Darcs is a free, open source revision control system. It is:

- Distributed: Every user has access to the full command set, removing boundaries between server and client or committer and non-committers.
- Interactive: Darcs is easy to learn and efficient to use because it asks you questions in response to simple commands, giving you choices in your work flow. You can choose to record one change in a file, while ignoring another. As you update from upstream, you can review each patch name, even the full 'diff' for interesting patches.
- Smart: Originally developed by physicist David Roundy, darcs is based on a unique algebra of patches. This smartness lets you respond to changing demands in ways that would otherwise not be possible. Learn more about spontaneous branches with darcs.

Below is a description of each darcs command.

Changing and querying the working copy:

add [OPTION]... <FILE or DIRECTORY> ...

Generally a repository contains both files that should be version controlled (such as source code) and files that Darcs should ignore (such as executables compiled from the source code). The 'darcs add' command is used to tell Darcs which files to version control.

When an existing project is first imported into a Darcs repository, it is common to run 'darcs add -r *' or 'darcs record -l' to add all initial source files into darcs.

Adding symbolic links (symlinks) is not supported.

Darcs will ignore all files and folders that look 'boring'. The --boring option overrides this behaviour.

Darcs will not add file if another file in the same folder has the same name, except for case. The --case-ok option overrides this behaviour. Windows and OS X usually use filesystems that do not allow files a folder to have the same name except for case (for example, 'ReadMe' and 'README'). If --case-ok is used, the repository might be unusable on those systems!

The --date-trick option allows you to enable an experimental trick to make add conflicts, in which two users each add a file or directory with the same name, less problematic. While this trick is completely safe, it is not clear to what extent it is beneficial.

remove [OPTION]... <FILE or DIRECTORY> ...

Remove should be called when you want to remove a file from your project, but don't actually want to delete the file. Otherwise just delete the file or directory, and darcs will notice that it has been removed. Be aware that the file WILL be deleted from any other copy of the repository to which you later apply the patch.

mv [OPTION]... [FILE or DIRECTORY]...

Darcs mv needs to be called whenever you want to move files or directories. Unlike remove, mv actually performs the move itself in your working copy.

replace [OPTION]... <OLD> <NEW> <FILE> ...

Replace allows you to change a specified token wherever it occurs in the specified files. The replace is encoded in a special patch and will merge as expected with other patches. Tokens here are defined by a regexp specifying the characters which are allowed. By default a token corresponds to a C identifier.

revert [OPTION]... [FILE or DIRECTORY]...

Revert is used to undo changes made to the working copy which have not yet been recorded. You will be prompted for which changes you wish to undo. The last revert can be undone safely using the unrevert command if the working copy was not modified in the meantime.

unrevert [OPTION]...

Unrevert is a rescue command in case you accidentally reverted something you wanted to keep (for example, accidentally typing ‘darcs rev -a’ instead of ‘darcs rec -a’).

This command may fail if the repository has changed since the revert took place. Darcs will ask for confirmation before executing an interactive command that will *definitely* prevent unreversion.

whatsnew [OPTION]... [FILE or DIRECTORY]...

The ‘darcs whatsnew’ command lists unrecorded changes to the working tree. If you specify a set of files and directories, only unrecorded changes to those files and directories are listed.

With the --summary option, the changes are condensed to one line per file, with mnemonics to indicate the nature and extent of the change. The --look-for-adds option causes candidates for ‘darcs add’ to be included in the summary output.

By default, ‘darcs whatsnew’ uses Darcs’ internal format for changes. To see some context (unchanged lines) around each change, use the --unified option. To view changes in conventional ‘diff’ format, use the ‘darcs diff’ comand; but note that ‘darcs whatsnew’ is faster.

This command exits unsuccessfully (returns a non-zero exit status) if there are no unrecorded changes.

Copying changes between the working copy and the repository:

record [OPTION]... [FILE or DIRECTORY]...

Record is used to name a set of changes and record the patch to the repository.

unrecord [OPTION]...

Unrecord does the opposite of record in that it makes the changes from patches active changes again which you may record or revert later. The working copy itself will not change. Beware that you should not use this command if you are going to re-record the changes in any way and there is a possibility that another user may have already pulled the patch.

amend-record [OPTION]... [FILE or DIRECTORY]...

Amend-record updates a ‘draft’ patch with additions or improvements, resulting in a single ‘finished’ patch. This is better than recording the additions and improvements as separate patches, because then whenever the ‘draft’ patch is copied between repositories, you would need to make sure all the extra patches are copied, too.

Do not copy draft patches between repositories, because a finished patch cannot be copied into a repository that contains a draft of the same patch. If this has already happened, ‘darcs obliterate’ can be used to remove the draft patch.

Do not run amend-record in repository that other developers can pull from, because if they pull while an amend-record is in progress, their repository may be corrupted.

When recording a draft patch, it is a good idea to start the name with ‘DRAFT:’ so that other developers know it is not finished. When finished, remove it with ‘darcs amend-record --edit-description’.

Like ‘darcs record’, if you call amend-record with files as arguments, you will only be asked about changes to those files. So to amend a patch to foo.c with improvements in bar.c, you would run:

darcs amend-record --match ’touch foo.c’ bar.c

It is usually a bad idea to amend another developer’s patch. To make amend-record only ask about your own patches by default, you can add something like ‘amend-record match David Roundy’ to ˜/.darcs/defaults, where ‘David Roundy’ is your name.

mark-conflicts [OPTION]...

Darcs requires human guidance to unify changes to the same part of a source file. When a conflict first occurs, darcs will add both choices to the working tree, delimited by markers.

However, you might revert or manually delete these markers without actually resolving the conflict. In this case, ‘darcs mark-conflicts’ is useful to show where any unresolved conflicts. It is also useful if ‘darcs apply’ is called with --apply-conflicts, where conflicts aren’t marked initially.

Any unrecorded changes to the working tree *will* be lost forever when you run this command! You will be prompted for confirmation before this takes place.

This command was historically called ‘resolve’, and this deprecated alias still exists for backwards-compatibility.

Direct modification of the repository:

tag [OPTION]... [TAGNAME]

The ‘darcs tag’ command names the current repository state, so that it can easily be referred to later. Every ‘important’ state should be tagged; in particular it is good practice to tag each stable release with a number or codename. Advice on release numbering can be found at http://producingoss.com/en/development-cycle.html.

To reproduce the state of a repository ‘R’ as at tag ‘t’, use the command ‘darcs get --tag t R’. The command ‘darcs show tags’ lists all tags in the current repository.

Tagging also provides significant performance benefits: when Darcs reaches a shared tag that depends on all antecedent patches, it can simply stop processing.

Like normal patches, a tag has a name, an author, a timestamp and an optional long description, but it does not change the working tree. A tag can have any name, but it is generally best to pick a naming scheme and stick to it.

The ‘darcs tag’ command accepts the --pipe and --checkpoint options, which behave as described in

‘darcs record’ and ‘darcs optimize’ respectively.

setpref [OPTION]... <PREF> <VALUE>

When working on project with multiple repositories and contributors, it is sometimes desirable for a preference to be set consistently project-wide. This is achieved by treating a preference set with ‘darcs setpref’ as an unrecorded change, which can then be recorded and then treated like any other patch.

Valid preferences are:

```
test -- a shell command that runs regression tests
predist -- a shell command to run before ‘darcs dist’
boringfile -- the path to a version-controlled boring file
binariesfile -- the path to a version-controlled binaries file
```

For example, a project using GNU autotools, with a ‘make test’ target to perform regression tests, might enable Darcs’ integrated regression testing with the following command:

```
darcs setpref test 'autoconf && ./configure && make && make test'
```

Note that merging is not currently implemented for preferences: if two patches attempt to set the same preference, the last patch applied to the repository will always take precedence. This is considered a low-priority bug, because preferences are seldom set.

Querying the repository:

diff [OPTION]... [FILE or DIRECTORY]...

Diff can be used to create a diff between two versions which are in your repository. Specifying just --from-patch will get you a diff against your working copy. If you give diff no version arguments, it gives you the same information as whatsnew except that the patch is formatted as the output of a diff command

changes [OPTION]... [FILE or DIRECTORY]...

Changes gives a changelog-style summary of the repository history, including options for altering how the patches are selected and displayed.

annotate [OPTION]... [FILE or DIRECTORY]...

Annotate displays which patches created or last modified a directory file or line. It can also display the contents of a particular patch in darcs format.

dist [OPTION]...

The ‘darcs dist’ command creates a compressed archive (a ‘tarball’) in the repository’s root directory, containing the recorded state of the working tree (unrecorded changes and the _darcs directory are excluded).

If a predist command is set (see ‘darcs setpref’), that command will be run on the tarball contents prior to archiving. For example, autotools projects would set it to ‘autoconf automake’.

By default, the tarball (and the top-level directory within the tarball) has the same name as the repository, but this can be overridden with the --dist-name option.

trackdown [OPTION]... [[INITIALIZATION] COMMAND]

Trackdown tries to find the most recent version in the repository which passes a test. Given no arguments, it uses the default repository test. Given one argument, it treats it as a test command. Given two arguments, the first is an initialization command with is run only once, and the second is the test command.

show contents [OPTION]... [FILE]...

Show contents can be used to display an earlier version of some file(s). If you give show contents no version arguments, it displays the recorded version of the file(s).

show files [OPTION]...

The files command lists the version-controlled files in the working copy. The similar manifest command, lists the same files, excluding any directories.

show repo [OPTION]...

The repo command displays information about the current repository (location, type, etc.). Some of this information is already available by inspecting files within the _darcs directory and some is internal information that is informational only (i.e. for developers). This command collects all of the repository information into a readily available source.

show authors [OPTION]...

The ‘darcs show authors’ command lists the authors of the current repository, sorted by the number of patches contributed. With the --verbose option, this command simply lists the author of each patch (without aggregation or sorting).

show tags [OPTION]...

The tags command writes a list of all tags in the repository to standard output.

Copying patches between repositories with working copy update:

pull [OPTION]... [REPOSITORY]...

Pull is used to bring changes made in another repository into the current repository (that is, either the one in the current directory, or the one specified with the --repodir option). Pull allows you to bring over all or some of the patches that are in that repository but not in this one. Pull accepts arguments, which are URLs from which to pull, and when called without an argument, pull will use the repository from which you have most recently either pushed or pulled.

obliterate [OPTION]...

Obliterate completely removes recorded patches from your local repository. The changes will be undone in your working copy and the patches will not be shown in your changes list anymore. Beware that you can lose precious code by obliterating!

rollback [OPTION]... [FILE or DIRECTORY]...

Rollback is used to undo the effects of one or more patches without actually deleting them. Instead, it creates a new patch reversing selected portions. of those changes. Unlike obliterate and unrecord (which accomplish a similar goal) rollback is perfectly safe, since it leaves in the repository a record of its changes.

push [OPTION]... [REPOSITORY]

Push is the opposite of pull. Push allows you to copy changes from the current repository into another repository.

send [OPTION]... [REPOSITORY]

Send is used to prepare a bundle of patches that can be applied to a target repository. Send accepts the URL of the repository as an argument. When called without an argument, send will use the most recent repository that was either pushed to, pulled from or sent to. By default, the patch bundle is sent by email, although you may save it to a file.

apply [OPTION]... <PATCHFILE>

Apply is used to apply a bundle of patches to this repository. Such a bundle may be created using send.

get [OPTION]... <REPOSITORY> [<DIRECTORY>]

Get creates a local copy of a repository. The optional second argument specifies a destination directory for the new copy; if omitted, it is inferred from the source location.

By default Darcs will copy every patch from the original repository. This means the copy is completely independent of the original; you can operate on the new repository even when the original is inaccessible. If you expect the original repository to remain accessible, you can use --lazy to avoid copying patches until they are needed ('copy on demand'). This is particularly useful when copying a remote repository with a long history that you don't care about.

The --lazy option isn't as useful for local copies, because Darcs will automatically use 'hard linking' where possible. As well as saving time and space, you can move or delete the original repository without affecting a complete, hard-linked copy. Hard linking requires that the copy be on the same filesystem and the original repository, and that the filesystem support hard linking. This is usually the case, except for Windows versions prior to Vista.

Darcs get will not copy unrecorded changes to the source repository's working tree.

It is often desirable to make a copy of a repository that excludes some patches. For example, if releases are tagged then 'darcs get --tag .' would make a copy of the repository as at the latest release.

An untagged repository state can still be identified unambiguously by a context file, as generated by 'darcs changes --context'. Given the name of such a file, the --context option will create a repository that includes only the patches from that context. When a user reports a bug in an unreleased version of your project, the recommended way to find out exactly what version they were running is to have them include a context file in the bug report.

You can also make a copy of an untagged state using the --to-patch or --to-match options, which exclude patches 'after' the first matching patch. Because these options treat the set of patches as an ordered sequence, you may get different results after reordering with 'darcs optimize', so tagging is preferred.

If the source repository is in a legacy darcs-1 format and contains at least one checkpoint (see ‘darcs optimize), the --partial option will create a partial repository. A partial repository discards history from before the checkpoint in order to reduce resource requirements. For modern darcs-2 repositories, --partial is a deprecated alias for the --lazy option.

put [OPTION]... <NEW REPOSITORY>

The ‘darcs put’ command creates a copy of the current repository. It is currently very inefficient, so when creating local copies you should use ‘darcs get . x’ instead of ‘darcs put x’.

Currently this command just uses ‘darcs init’ to create the target repository, then ‘darcs push --all’ to copy patches to it. Options passed to ‘darcs put’ are passed to the init and/or push commands as appropriate. See those commands for an explanation of each option.

Administrating repositories:

initialize [OPTION]...

The ‘darcs initialize’ command turns the current directory into a Darcs repository. Any existing files and subdirectories become UNSAVED changes in the working tree: record them with ‘darcs add -r’ and ‘darcs record’.

When converting a project to Darcs from some other VCS, translating the full revision history to native Darcs patches is recommended. (The Darcs wiki lists utilities for this.) Because Darcs is optimized for small patches, simply importing the latest revision as a single large patch can PERMANENTLY degrade Darcs performance in your repository by an order of magnitude.

This command creates the ‘_darcs’ directory, which stores version control metadata. It also contains per-repository settings in _darcs/prefs/, which you can read about in the user manual.

In addition to the default darcs-2 format, there are two backwards-compatible formats for the _darcs directory. If all contributors to your project have darcs 2.0.0 or higher, use the default format.

If some contributors still run Darcs below 2.0.0, you need to use the ‘old-fashioned inventory’ format for any repositories those contributors access. Because patches cannot be shared between darcs-2 and old-fashioned repositories, other project repos should use the intermediary ‘hashed’ format.

Darcs will create a hashed repository by default when you ‘darcs get’ a repository in old-fashioned inventory format. Once all contributors have upgraded to Darcs 2.0.0 or later, use ‘darcs convert’ to convert the project to the darcs-2 format.

Initialize is commonly abbreviated to ‘init’.

optimize [OPTION]...

Optimize can help to improve the performance of your repository in a number of cases.

check [OPTION]...

This command verifies that the patches in the repository, when applied successively to an empty tree, result in the pristine tree. If not, the differences are printed and Darcs exits unsucessfully (with a non-zero exit status).

If the repository is in darcs-1 format and has a checkpoint, you can use the --partial option to start checking from the latest checkpoint. This is the default for partial darcs-1 repositories; the --complete

option to forces a full check.

If a regression test is defined (see ‘darcs setpref’) it will be run by ‘darcs check’. Use the --no-test option to disable this.

repair [OPTION]...
The ‘darcs repair’ command attempts to fix corruption in the current repository. Currently it can only repair damage to the pristine tree, which is where most corruption occurs.

convert [OPTION]... <REPOSITORY> [<DIRECTORY>]
Convert is used to convert a repository to darcs-2 format.

The recommended way to convert an existing project from darcs 1 to darcs 2 is to merge all branches, ‘darcs convert’ the resulting repository, re-create each branch by using ‘darcs get’ on the converted repository, then using ‘darcs obliterate’ to delete patches of branches.

BUGS

At http://bugs.darcs.net/ you can find a list of known bugs in Darcs. Unknown bugs can be reported at that site (after creating an account) or by emailing the report to bugs@darcs.net.

SEE ALSO

A user manual is included with Darcs, in PDF and HTML form. It can also be found at http://darcs.net/manual/.