git-svn

- config key: svn-remote.<name>.ignore-paths If the ignore-paths configuration key is set, and the com
- git rebase --onto remotes/git-svn A^ master ---------- r1---r2'--r3’ remotes/git-svn \ A'--B’ mast

---

git-svn(1) ==========

NAME ---- git-svn - Bidirectional operation between a Subversion repository and Git

SYNOPSIS --------

> ’git svn’ <command> [<options>] [<arguments>]

DESCRIPTION ----------- ’git svn’ is a simple conduit for changesets between Subversion and Git. It provides a bidirectional flow of changes between a Subversion and a Git repository.

’git svn’ can track a standard Subversion repository, following the common "trunk/branches/tags" layout, with the --stdlayout option. It can also follow branches and tags in any layout with the -T/-t/-b options (see options to ’init’ below, and also the ’clone’ command).

Once tracking a Subversion repository (with any of the above methods), the Git repository can be updated from Subversion by the ’fetch’ command and Subversion updated from Git by the ’dcommit’ command.

COMMANDS --------

’init’
Initializes an empty Git repository with additional metadata directories for ’git svn’. The Subversion URL may be specified as a command-line argument, or as full URL arguments to -T/-t/-b. Optionally, the target directory to operate on can be specified as a second argument. Normally this command initializes the current directory.

-T<trunk-subdir>;; --trunk=<trunk-subdir>;; -t<tags-subdir>;; --tags=<tags-subdir>;; -b<branches-subdir>;; --branches=<branches-subdir>;; -s;; --stdlayout;; These are optional command-line options for init. Each of these flags can point to a relative repository path (--

tags=project/tags) or a full url (--tags=https://foo.org/project/tags). You can specify more than one --tags and/or --branches options, in case your Subversion repository places tags or branches under multiple paths. The option --stdlayout is a shorthand way of setting trunk,tags,branches as the relative paths, which is the Subversion default. If any of the other options are given as well, they take precedence. --no-metadata;; Set the 'noMetadata' option in the [svn-remote] config. This option is not recommended, please read the 'svn.noMetadata' section of this manpage before using this option. --use-svm-props;; Set the 'useSvmProps' option in the [svn-remote] config. --use-svnsync-props;; Set the 'useSvnsyncProps' option in the [svn-remote] config. --rewrite-root=<URL>;; Set the 'rewriteRoot' option in the [svn-remote] config. --rewrite-uuid=<UUID>;; Set the 'rewriteUUID' option in the [svn-remote] config. --username=<user>;; For transports that SVN handles authentication for (http, https, and plain svn), specify the username. For other transports (e.g. `svn+ssh://`), you must include the username in the URL, e.g. `svn+ssh://foo@svn.bar.com/project` --prefix=<prefix>;; This allows one to specify a prefix which is prepended to the names of remotes if trunk/branches/tags are specified. The prefix does not automatically include a trailing slash, so be sure you include one in the argument if that is what you want. If --branches/-b is specified, the prefix must include a trailing slash. Setting a prefix (with a trailing slash) is strongly encouraged in any case, as your SVN-tracking refs will then be located at "refs/remotes/$prefix/**", which is compatible with Git's own remote-tracking ref layout (refs/remotes/$remote/**). Setting a prefix is also useful if you wish to track multiple projects that share a common repository. By default, the prefix is set to 'origin/'.
NOTE: Before Git v2.0, the default prefix was "" (no prefix). This meant that SVN-tracking refs were put at "refs/remotes/*", which is incompatible with how Git's own remote-tracking refs are organized. If you still want the old default, you can get it by passing `--prefix ""` on the command line (`--prefix=""` may not work if your Perl's Getopt::Long is < v2.37).

--ignore-refs=<regex>;; When passed to 'init' or 'clone' this regular expression will be preserved as a config key. See 'fetch' for a description of `--ignore-refs`. --ignore-paths=<regex>;; When passed to 'init' or 'clone' this regular expression will be preserved as a config key. See 'fetch' for a description of `--ignore-paths`. --include-paths=<regex>;; When passed to 'init' or 'clone' this regular expression will be preserved as a config key. See 'fetch' for a description of `--include-paths`. --no-minimize-url;; When tracking multiple directories (using --stdlayout, --branches, or --tags options), git svn will attempt to connect to the root (or highest allowed level) of the Subversion repository. This default allows better tracking of history if entire projects are moved within a repository, but may cause issues on repositories where read access restrictions are in place. Passing `--no-minimize-url` will allow git svn to accept URLs as-is without attempting to connect to a higher level directory. This option is off by default when only one URL/branch is tracked (it would do little good).

'fetch'
Fetch unfetched revisions from the Subversion remote we are tracking. The name of the [svn-remote "..."] section in the $GIT_DIR/config file may be specified as an optional command-line argument.

This automatically updates the rev_map if needed (see '$GIT_DIR/svn/***/.rev_map.**' in the FILES section below for details).

--localtime;; Store Git commit times in the local time zone instead of UTC. This makes 'git log' (even without --date=local) show the same times that `svn log` would in the local time zone. This doesn't interfere with interoperating with the Subversion repository you cloned from, but if you wish for your local Git repository to be able to interoperate with someone else's local Git repository, either don't use this option or you should both use it in the same local time zone.

--parent;; Fetch only from the SVN parent of the current HEAD.

--ignore-refs=<regex>;; Ignore refs for branches or tags matching the Perl regular expression. A "negative look-ahead assertion" like `^refs/remotes/origin/(?!tags/wanted-tag|wanted-branch).*$` can be used to allow only certain refs. +

> config key: svn-remote.<name>.ignore-refs
> If the ignore-refs configuration key is set, and the command-line
> option is also given, both regular expressions will be used.

--ignore-paths=<regex>;; This allows one to specify a Perl regular expression that will cause skipping of all matching paths from checkout from SVN. The `--ignore-paths` option should match for every 'fetch' (including automatic fetches due to 'clone', 'dcommit', 'rebase', etc) on a given repository. +

> "
>
> config key: svn-remote.<name>.ignore-paths
> If the ignore-paths
> configuration key is set, and the command-line
> option is also given, both

> regular expressions will be used.
> Examples:
> --
> Skip "doc*" directory for every fetch;;
> ----------
> ----------
> --ignore-paths="^doc"\

Skip "branches" and "tags" of first level directories;; ---------- --ignore-paths="[/]+/(?:branches|tags)" ---------- --

--include-paths=<regex>;; This allows one to specify a Perl regular expression that will cause the inclusion of only matching paths from checkout from SVN. The `--include-paths` option should match for every ’fetch’ (including automatic fetches due to ’clone’, ’dcommit’, ’rebase’, etc) on a given repository. `--ignore-paths` takes precedence over `--include-paths`. +

> config key: svn-remote.<name>.include-paths

--log-window-size=<n>;; Fetch <n> log entries per request when scanning Subversion history. The default is 100. For very large Subversion repositories, larger values may be needed for ’clone’/’fetch’ to complete in reasonable time. But overly large values may lead to higher memory usage and request timeouts.

’clone’
Runs ’init’ and ’fetch’. It will automatically create a directory based on the basename of the URL passed to it; or if a second argument is passed; it will create a directory and work within that. It accepts all arguments that the ’init’ and ’fetch’ commands accept; with the exception of `--fetch-all` and `--parent`. After a repository is cloned, the ’fetch’ command will be able to update revisions without affecting the working tree; and the ’rebase’ command will be able to update the working tree with the latest changes.

--preserve-empty-dirs;; Create a placeholder file in the local Git repository for each empty directory fetched from Subversion. This includes directories that become empty by removing all entries in the Subversion repository (but not the directory itself). The placeholder files are also tracked and removed when no longer necessary.

--placeholder-filename=<filename>;; Set the name of placeholder files created by --preserve-empty-dirs. Default: ".gitignore"

’rebase’
This fetches revisions from the SVN parent of the current HEAD and rebases the current (uncommitted to SVN) work against it.

This works similarly to `svn` `update` or ’git pull’ except that it preserves linear history with ’git rebase’ instead of ’git merge’ for ease of dcommitting with ’git svn’.

This accepts all options that ’git svn fetch’ and ’git rebase’ accept. However, `--fetch-all` only fetches from the current [svn-remote], and not all [svn-remote] definitions.

Like ’git rebase’; this requires that the working tree be clean and have no uncommitted changes.

This automatically updates the rev_map if needed (see ’$GIT_DIR/svn/***/.rev_map.**’ in the FILES section below for details).

-l;; --local;; Do not fetch remotely; only run ’git rebase’ against the last fetched commit from the upstream SVN.

’dcommit’
Commit each diff from the current branch directly to the SVN repository, and then rebase or reset (depending on whether or not there is a diff between SVN and head). This will create a revision in SVN for each commit in Git.

When an optional Git branch name (or a Git commit object name) is specified as an argument, the subcommand works on the specified branch, not on the current branch.

Use of ’dcommit’ is preferred to ’set-tree’ (below).

--no-rebase;; After committing, do not rebase or reset. --commit-url <URL>;; Commit to this SVN URL (the full path). This is intended to allow existing ’git svn’ repositories created with one transport method (e.g. `svn://` or

`http://` for anonymous read) to be reused if a user is later given access to an alternate transport method (e.g. `svn+ssh://` or `https://`) for commit.

> config key: svn-remote.<name>.commiturl
> config key: svn.commiturl (overwrites all svn-remote.<name>.commiturl options)

Note that the SVN URL of the commiturl config key includes the SVN branch. If you rather want to set the commit URL for an entire SVN repository use svn-remote.<name>.pushurl instead.

Using this option for any other purpose (don't ask) is very strongly discouraged.

--mergeinfo=<mergeinfo>;; Add the given merge information during the dcommit (e.g. `--mergeinfo="/branches/foo:1-10"`). All svn server versions can store this information (as a property), and svn clients starting from version 1.5 can make use of it. To specify merge information from multiple branches, use a single space character between the branches (`--mergeinfo="/branches/foo:1-10 /branches/bar:3,5-6,8"`) +

> config key: svn.pushmergeinfo
> This option will cause git-svn to attempt to automatically populate the svn:mergeinfo property in the SVN repository when possible. Currently, this can only be done when dcommitting non-fast-forward merges where all parents but the
> first have already been pushed into SVN.

--interactive;; Ask the user to confirm that a patch set should actually be sent to SVN. For each patch, one may answer "yes" (accept this patch), "no" (discard this patch), "all" (accept all patches), or "quit".
'git svn dcommit' returns immediately if answer is "no" or "quit", without committing anything to SVN.

'branch'
Create a branch in the SVN repository.

-m;; --message;; Allows to specify the commit message.

-t;; --tag;; Create a tag by using the tags_subdir instead of the branches_subdir specified during git svn init.

-d<path>;; --destination=<path>;;

```
If more than one --branches (or --tags) option was given to the 'init'
or 'clone' command, you must provide the location of the branch (or
tag) you wish to create in the SVN repository.  <path> specifies which
path to use to create the branch or tag and should match the pattern
on the left-hand side of one of the configured branches or tags
refspecs.  You can see these refspecs with the commands
```

git config --get-all svn-remote.<name>.branches git config --get-all svn-remote.<name>.tags where <name> is the name of the SVN repository as specified by the -R option to 'init' (or "svn" by default).

--username;; Specify the SVN username to perform the commit as. This option overrides the 'username' configuration property.

--commit-url;; Use the specified URL to connect to the destination Subversion repository. This is useful in cases where the source SVN repository is read-only. This option overrides configuration property 'commiturl'.
git config --get-all svn-remote.<name>.commiturl +

--parents;; Create parent folders. This parameter is equivalent to the parameter --parents on svn cp commands and is useful for non-standard repository layouts.

'tag'
Create a tag in the SVN repository. This is a shorthand for 'branch -t'.

'log'
This should make it easy to look up svn log messages when svn users refer to -r/--revision numbers.

The following features from `svn log' are supported:

-r <n>[:<n>];; --revision=<n>[:<n>];; is supported, non-numeric args are not: HEAD, NEXT, BASE, PREV, etc ... -v;; --verbose;; it's not completely compatible with the --verbose output in svn log, but reasonably close. --limit=<n>;; is NOT the same as --max-count, doesn't count merged/excluded commits --incremental;; supported

New features:

--show-commit;; shows the Git commit sha1, as well --oneline;; our version of --pretty=oneline

> [!NOTE] SVN itself only stores times in UTC and nothing else. The regular svn client converts the UTC time to the local time (or based on the TZ=

> environment). This command has the same behaviour.

Any other arguments are passed directly to ’git log’

’blame’
Show what revision and author last modified each line of a file. The output of this mode is format-compatible with the output of `svn blame’ by default. Like the SVN blame command, local uncommitted changes in the working tree are ignored; the version of the file in the HEAD revision is annotated. Unknown arguments are passed directly to ’git blame’.

--git-format;; Produce output in the same format as ’git blame’, but with SVN revision numbers instead of Git commit hashes. In this mode, changes that haven’t been committed to SVN (including local working-copy edits) are shown as revision 0.

’find-rev’
When given an SVN revision number of the form ’rN’, returns the corresponding Git commit hash (this can optionally be followed by a tree-ish to specify which branch should be searched). When given a tree-ish, returns the corresponding SVN revision number.

-B;; --before;; Don’t require an exact match if given an SVN revision, instead find the commit corresponding to the state of the SVN repository (on the current branch) at the specified revision.

-A;; --after;; Don’t require an exact match if given an SVN revision; if there is not an exact match return the closest match searching forward in the history.

’set-tree’
You should consider using ’dcommit’ instead of this command. Commit specified commit or tree objects to SVN. This relies on your imported fetch data being up to date. This makes absolutely no attempts to do patching when committing to SVN, it simply overwrites files with those specified in the tree or commit. All merging is assumed to have taken place independently of ’git svn’ functions.

’create-ignore’
Recursively finds the svn:ignore and svn:global-ignores properties on directories and creates matching .gitignore files. The resulting files are staged to be committed, but are not committed. Use -r/--revision to refer to a specific revision.

’show-ignore’
Recursively finds and lists the svn:ignore and svn:global-ignores properties on directories. The output is suitable for appending to the $GIT_COMMON_DIR/info/exclude file.

’mkdirs’
Attempts to recreate empty directories that core Git cannot track based on information in $GIT_DIR/svn/<refname>/unhandled.log files. Empty directories are automatically recreated when using "git svn clone" and "git svn rebase", so "mkdirs" is intended for use after commands like "git

checkout" or "git reset". (See the svn-remote.<name>.automkdirs config file option for more information.)

’commit-diff’
Commits the diff of two tree-ish arguments from the command-line. This command does not rely on being inside a `git` `svn` `init`-ed repository. This command takes three arguments, (a) the original tree to diff against, (b) the new tree result, (c) the URL of the target Subversion repository. The final argument (URL) may be omitted if you are working from a ’git svn’-aware repository (that has been `init`-ed with ’git svn’). The -r<revision> option is required for this.

The commit message is supplied either directly with the `-m` or `-F` option, or indirectly from the tag or commit when the second tree-ish denotes such an object, or it is requested by invoking an editor (see `--edit` option below).

-m <msg>;; --message=<msg>;; Use the given `msg` as the commit message. This option disables the `--edit` option.

-F <filename>;; --file=<filename>;; Take the commit message from the given file. This option disables the `--edit` option.

’info’
Shows information about a file or directory similar to what `svn info’ provides. Does not currently support a -r/--revision argument. Use the --url option to output only the value of the ’URL:’ field.

’proplist’
Lists the properties stored in the Subversion repository about a given file or directory. Use -r/--revision to refer to a specific Subversion revision.

’propget’
Gets the Subversion property given as the first argument, for a file. A specific revision can be specified with -r/--revision.

’propset’
Sets the Subversion property given as the first argument, to the value given as the second argument for the file given as the third argument.

Example:

```
git svn propset svn:keywords "FreeBSD=%H" devel/py-tipper/Makefile
```

This will set the property ’svn:keywords’ to ’FreeBSD=%H’ for the file ’devel/py-tipper/Makefile’.

’show-externals’
Shows the Subversion externals. Use -r/--revision to specify a specific revision.

’gc’
Compress $GIT_DIR/svn/<refname>/unhandled.log files and remove

$GIT_DIR/svn/<refname>/index files.

’reset’
Undoes the effects of ’fetch’ back to the specified revision. This allows you to re-’fetch’ an SVN revision. Normally the contents of an SVN revision should never change and ’reset’ should not be necessary. However, if SVN permissions change, or if you alter your --ignore-paths option, a ’fetch’ may fail with "not found in commit" (file not previously visible) or "checksum mismatch" (missed a modification). If the problem file cannot be ignored forever (with --ignore-paths) the only way to repair the repo is to use ’reset’.

Only the rev_map and refs/remotes/git-svn are changed (see ’$GIT_DIR/svn/***/.rev_map.**’ in the FILES section below for details). Follow ’reset’ with a ’fetch’ and then ’git reset’ or ’git rebase’ to move local branches onto the new tree.

-r <n>;; --revision=<n>;; Specify the most recent revision to keep. All later revisions are discarded. -p;; --parent;; Discard the specified revision as well, keeping the nearest parent instead. Example:;; Assume you have local changes in "master", but you need to refetch "r2".
---------- r1---r2---r3 remotes/git-svn \ A---B master ----------
Fix the ignore-paths or SVN permissions problem that caused "r2" to be incomplete in the first place. Then: +

> “ git svn reset -r2 -p
> git svn fetch
> ----------
> r1---r2'--r3’ remotes/git-svn
> \
> r2---r3---A---B master
> ----------
> Then fixup "master" with ’git rebase’.
> Do NOT use ’git merge’ or your history will not be compatible with a future ’dcommit’!
> +

> “
>
> git rebase --onto remotes/git-svn A^ master

> ----------
> r1---r2'--r3’ remotes/git-svn
> \
> A'--B’ master\

OPTIONS -------

--shared[=(false|true|umask|group|all|world|everybody)]

--template=<template-directory>
Only used with the ’init’ command. These are passed directly to ’git init’.

-r <arg>

--revision <arg>
Used with the ’fetch’ command.

This allows revision ranges for partial/cauterized history to be supported. $NUMBER, $NUMBER1:$NUMBER2 (numeric ranges), $NUMBER:HEAD, and BASE:$NUMBER are all supported.

This can allow you to make partial mirrors when running fetch; but is generally not recommended because history will be skipped and lost.

-

--stdin
Only used with the ’set-tree’ command.

Read a list of commits from stdin and commit them in reverse order. Only the leading sha1 is read from each line, so ’git rev-list --pretty=oneline’ output can be used.

--rmdir
Only used with the ’dcommit’, ’set-tree’ and ’commit-diff’ commands.

Remove directories from the SVN tree if there are no files left behind. SVN can version empty directories, and they are not removed by default if there are no files left in them. Git cannot version empty directories. Enabling this flag will make the commit to SVN act like Git.

>

> config key: svn.rmdir

-e

--edit
Only used with the ’dcommit’, ’set-tree’ and ’commit-diff’ commands.

Edit the commit message before committing to SVN. This is off by default for objects that are commits, and forced on when committing tree objects.

> config key: svn.edit

-l<num>

--find-copies-harder
Only used with the ’dcommit’, ’set-tree’ and ’commit-diff’ commands.

They are both passed directly to ’git diff-tree’; see linkgit:git-diff-tree[1] for more information.

> config key: svn.l
> config key: svn.findcopiesharder

-A<filename>

--authors-file=<filename>
Syntax is compatible with the file used by ’git cvsimport’ but an empty email address can be supplied with ’<>’:

```
loginname = Joe User <user@example.com>
```

If this option is specified and ’git svn’ encounters an SVN committer name that does not exist in the authors-file, ’git svn’ will abort operation. The user will then have to add the appropriate entry. Re-running the previous ’git svn’ command after the authors-file is modified should continue operation.

> config key: svn.authorsfile

--authors-prog=<filename>
If this option is specified, for each SVN committer name that does not exist in the authors file, the

given file is executed with the committer name as the first argument. The program is expected to return a single line of the form "Name <email>" or "Name <>", which will be treated as if included in the authors file.

Due to historical reasons a relative ’filename’ is first searched relative to the current directory for ’init’ and ’clone’ and relative to the root of the working tree for ’fetch’. If ’filename’ is not found, it is searched like any other command in ’$PATH’.

> config key: svn.authorsProg

-q

--quiet
Make ’git svn’ less verbose. Specify a second time to make it even less verbose.

-m

--merge

-s<strategy>

--strategy=<strategy>

-p

--rebase-merges
These are only used with the ’dcommit’ and ’rebase’ commands.

Passed directly to ’git rebase’ when using ’dcommit’ if a ’git reset’ cannot be used (see ’dcommit’).

-n

--dry-run
This can be used with the ’dcommit’, ’rebase’, ’branch’ and ’tag’ commands.

For ’dcommit’, print out the series of Git arguments that would show which diffs would be committed to SVN.

For ’rebase’, display the local branch associated with the upstream svn repository associated with the current branch and the URL of svn repository that will be fetched from.

For ’branch’ and ’tag’, display the urls that will be used for copying when creating the branch or tag.

--use-log-author
When retrieving svn commits into Git (as part of ’fetch’, ’rebase’, or ’dcommit’ operations), look for the first `From:` line or `Signed-off-by` trailer in the log message and use that as the author string.

> config key: svn.useLogAuthor

--add-author-from
When committing to svn from Git (as part of ’set-tree’ or ’dcommit’ operations), if the existing log message doesn’t already have a `From:` or `Signed-off-by` trailer, append a `From:` line based on the Git commit’s author string. If you use this, then `--use-log-author` will retrieve a valid author string for all commits.

> config key: svn.addAuthorFrom

ADVANCED OPTIONS ----------

-i<GIT_SVN_ID>

--id <GIT_SVN_ID>
This sets GIT_SVN_ID (instead of using the environment). This allows the user to override the default refname to fetch from when tracking a single URL. The ’log’ and ’dcommit’ commands no longer require this switch as an argument.

-R<remote-name>

--svn-remote <remote-name>
Specify the [svn-remote "<remote-name>"] section to use, this allows SVN multiple repositories to be tracked. Default: "svn"

--follow-parent
This option is only relevant if we are tracking branches (using one of the repository layout options --trunk, --tags, --branches, --stdlayout). For each tracked branch, try to find out where its revision was copied from, and set a suitable parent in the first Git commit for the branch. This is especially helpful when we’re tracking a directory that has been moved around within the repository. If this feature is disabled, the branches created by ’git svn’ will all be linear and not share any history, meaning that there will be no information on where branches were branched off or merged. However, following long/convoluted histories can take a long time, so disabling this feature may speed up the cloning process. This feature is enabled by default, use --no-follow-parent to disable it.

>

> config key: svn.followparent

CONFIG FILE-ONLY OPTIONS ----------

svn.noMetadata

svn-remote.<name>.noMetadata
This gets rid of the ’git-svn-id:’ lines at the end of every commit.

This option can only be used for one-shot imports as ’git svn’ will not be able to fetch again without metadata. Additionally, if you lose your ’$GIT_DIR/svn/***/.rev_map.**’ files, ’git svn’ will not be able to rebuild them.

The ’git svn log’ command will not work on repositories using this, either. Using this conflicts with the ’useSvmProps’ option for (hopefully) obvious reasons.

This option is NOT recommended as it makes it difficult to track down old references to SVN revision numbers in existing documentation, bug reports, and archives. If you plan to eventually migrate from SVN to Git and are certain about dropping SVN history, consider git-filter-repo instead. filter-repo also allows reformatting of metadata for ease-of-reading and rewriting authorship info for non-"svn.authorsFile" users.

svn.useSvmProps

svn-remote.<name>.useSvmProps
This allows ’git svn’ to re-map repository URLs and UUIDs from mirrors created using SVN::Mirror (or svk) for metadata.

If an SVN revision has a property, "svm:headrev", it is likely that the revision was created by SVN::Mirror (also used by SVK). The property contains a repository UUID and a revision. We want to make it look like we are mirroring the original URL, so introduce a helper function that returns the original identity URL and UUID, and use it when generating metadata in commit messages.

svn.useSvnsyncProps

svn-remote.<name>.useSvnsyncprops
Similar to the useSvmProps option; this is for users of the svnsync(1) command distributed with SVN 1.4.x and later.

svn-remote.<name>.rewriteRoot
This allows users to create repositories from alternate URLs. For example, an administrator could run ’git svn’ on the server locally (accessing via file://) but wish to distribute the repository with a public http:// or svn: metadata so users of it will see the public URL.

svn-remote.<name>.rewriteUUID
Similar to the useSvmProps option; this is for users who need to remap the UUID manually. This may be useful in situations where the original UUID is not available via either useSvmProps or useSvnsyncProps.

svn-remote.<name>.pushurl

```
Similar to Git's `remote.<name>.pushurl`, this key is designed
to be used in cases where 'url' points to an SVN repository
via a read-only transport, to provide an alternate read/write
transport. It is assumed that both keys point to the same
repository. Unlike 'commiturl', 'pushurl' is a base path. If
either 'commiturl' or 'pushurl' could be used, 'commiturl'
takes precedence.
```

svn.brokenSymlinkWorkaround
This disables potentially expensive checks to workaround broken symlinks checked into SVN by broken clients. Set this option to "false" if you track a SVN repository with many empty blobs that are not symlinks. This option may be changed while ’git svn’ is running and take effect on the next revision fetched. If unset, ’git svn’ assumes this option to be "true".

svn.pathnameencoding
This instructs git svn to recode pathnames to a given encoding. It can be used by windows users and by those who work in non-utf8 locales to avoid corrupted file names with non-ASCII characters. Valid encodings are the ones supported by Perl’s Encode module.

svn-remote.<name>.automkdirs
Normally, the "git svn clone" and "git svn rebase" commands attempt to recreate empty directories that are in the Subversion repository. If this option is set to "false", then empty directories will only be created if the "git svn mkdirs" command is run explicitly. If unset, ’git svn’ assumes this option to be "true".

Since the noMetadata, rewriteRoot, rewriteUUID, useSvnsyncProps and useSvmProps options all affect the metadata generated and used by ’git svn’; they **must** be set in the configuration file before any history is imported and these settings should never be changed once they are set.

Additionally, only one of these options can be used per svn-remote section because they affect the ’git-svn-id:’ metadata line, except for rewriteRoot and rewriteUUID which can be used together.

BASIC EXAMPLES ----------

Tracking and contributing to the trunk of a Subversion-managed project (ignoring tags and branches):

```
# Clone a repo (like git clone):
    git svn clone http://svn.example.com/project/trunk
# Enter the newly cloned directory:
    cd trunk
# You should be on master branch, double-check with 'git branch'
    git branch
# Do some work and commit locally to Git:
    git commit ...
# Something is committed to SVN, rebase your local changes against the
# latest changes in SVN:
    git svn rebase
# Now commit your changes (that were committed previously using Git) to SVN,
# as well as automatically updating your working HEAD:
    git svn dcommit
# Append svn:ignore and svn:global-ignores settings to the default Git exclude file:
    git svn show-ignore >> .git/info/exclude
```

Tracking and contributing to an entire Subversion-managed project (complete with a trunk, tags and branches):

```
# Clone a repo with standard SVN directory layout (like git clone):
    git svn clone http://svn.example.com/project --stdlayout --prefix svn/
# Or, if the repo uses a non-standard directory layout:
    git svn clone http://svn.example.com/project -T tr -b branch -t tag --prefix svn/
# View all branches and tags you have cloned:
    git branch -r
# Create a new branch in SVN
    git svn branch waldo
# Reset your master to trunk (or any other branch, replacing 'trunk'
# with the appropriate name):
    git reset --hard svn/trunk
# You may only dcommit to one branch/tag/trunk at a time.  The usage
# of dcommit/rebase/show-ignore should be the same as above.
```

The initial ’git svn clone’ can be quite time-consuming (especially for large Subversion repositories). If multiple people (or one person with multiple machines) want to use ’git svn’ to interact with the same Subversion repository, you can do the initial ’git svn clone’ to a repository on a server and have each person clone that repository with ’git clone’:

```
# Do the initial import on a server
    ssh server "cd /pub && git svn clone http://svn.example.com/project [options...]"
```

```
# Clone locally - make sure the refs/remotes/ space matches the server
    mkdir project
    cd project
    git init
    git remote add origin server:/pub/project
    git config --replace-all remote.origin.fetch '+refs/remotes/*:refs/remotes/*'
    git fetch
# Prevent fetch/pull from remote Git server in the future,
# we only want to use git svn for future updates
    git config --remove-section remote.origin
# Create a local branch from one of the branches just fetched
    git checkout -b master FETCH_HEAD
# Initialize 'git svn' locally (be sure to use the same URL and
# --stdlayout/-T/-b/-t/--prefix options as were used on server)
    git svn init http://svn.example.com/project [options...]
# Pull the latest changes from Subversion
    git svn rebase
```

REBASE VS. PULL/MERGE ---------- Prefer to use ’git svn rebase’ or ’git rebase’, rather than ’git pull’ or ’git merge’ to synchronize unintegrated commits with a ’git svn’ branch. Doing so will keep the history of unintegrated commits linear with respect to the upstream SVN repository and allow the use of the preferred ’git svn dcommit’ subcommand to push unintegrated commits back into SVN.

Originally, ’git svn’ recommended that developers pulled or merged from the ’git svn’ branch. This was because the author favored `git svn set-tree B` to commit a single head rather than the `git svn set-tree A..B` notation to commit multiple commits. Use of ’git pull’ or ’git merge’ with `git svn set-tree A..B` will cause non-linear history to be flattened when committing into SVN and this can lead to merge commits unexpectedly reversing previous commits in SVN.

MERGE TRACKING -------------- While ’git svn’ can track copy history (including branches and tags) for repositories adopting a standard layout, it cannot yet represent merge history that happened inside git back upstream to SVN users. Therefore it is advised that users keep history as linear as possible inside Git to ease compatibility with SVN (see the CAVEATS section below).

HANDLING OF SVN BRANCHES ---------- If ’git svn’ is configured to fetch branches (and --follow-branches is in effect), it sometimes creates multiple Git branches for one SVN branch, where the additional branches have names of the form ’branchname@nnn’ (with nnn an SVN revision number). These additional branches are created if ’git svn’ cannot find a parent commit for the first commit in an SVN branch, to connect the branch to the history of the other branches.

Normally, the first commit in an SVN branch consists of a copy operation. ’git svn’ will read this commit to get the SVN revision the branch was created from. It will then try to find the Git commit

that corresponds to this SVN revision, and use that as the parent of the branch. However, it is possible that there is no suitable Git commit to serve as parent. This will happen, among other reasons, if the SVN branch is a copy of a revision that was not fetched by 'git svn' (e.g. because it is an old revision that was skipped with `--revision`), or if in SVN a directory was copied that is not tracked by 'git svn' (such as a branch that is not tracked at all, or a subdirectory of a tracked branch). In these cases, 'git svn' will still create a Git branch, but instead of using an existing Git commit as the parent of the branch, it will read the SVN history of the directory the branch was copied from and create appropriate Git commits. This is indicated by the message "Initializing parent: <branchname>".

Additionally, it will create a special branch named '<branchname>@<SVN-Revision>', where <SVN-Revision> is the SVN revision number the branch was copied from. This branch will point to the newly created parent commit of the branch. If in SVN the branch was deleted and later recreated from a different version, there will be multiple such branches with an '@'.

Note that this may mean that multiple Git commits are created for a single SVN revision.

An example: in an SVN repository with a standard trunk/tags/branches layout, a directory trunk/sub is created in r.100. In r.200, trunk/sub is branched by copying it to branches/. 'git svn clone -s' will then create a branch 'sub'. It will also create new Git commits for r.100 through r.199 and use these as the history of branch 'sub'. Thus there will be two Git commits for each revision from r.100 to r.199 (one containing trunk/, one containing trunk/sub/). Finally, it will create a branch 'sub@200' pointing to the new parent commit of branch 'sub' (i.e. the commit for r.200 and trunk/sub/).

CAVEATS -------

For the sake of simplicity and interoperating with Subversion, it is recommended that all 'git svn' users clone, fetch and dcommit directly from the SVN server, and avoid all 'git clone'/'pull'/'merge'/'push' operations between Git repositories and branches. The recommended method of exchanging code between Git branches and users is 'git format-patch' and 'git am', or just 'dcommit'ing to the SVN repository.

Running 'git merge' or 'git pull' is NOT recommended on a branch you plan to 'dcommit' from because Subversion users cannot see any merges you've made. Furthermore, if you merge or pull from a Git branch that is a mirror of an SVN branch, 'dcommit' may commit to the wrong branch.

If you do merge, note the following rule: 'git svn dcommit' will attempt to commit on top of the SVN commit named in

```
git log --grep=^git-svn-id: --first-parent -1
```

You 'must' therefore ensure that the most recent commit of the branch you want to dcommit to is the 'first' parent of the merge. Chaos will ensue otherwise, especially if the first parent is an older commit on the same SVN branch.

’git clone’ does not clone branches under the refs/remotes/ hierarchy or any ’git svn’ metadata, or config. So repositories created and managed with using ’git svn’ should use ’rsync’ for cloning, if cloning is to be done at all.

Since ’dcommit’ uses rebase internally, any Git branches you ’git push’ to before ’dcommit’ on will require forcing an overwrite of the existing ref on the remote repository. This is generally considered bad practice, see the linkgit:git-push[1] documentation for details.

Do not use the --amend option of linkgit:git-commit[1] on a change you’ve already dcommitted. It is considered bad practice to --amend commits you’ve already pushed to a remote repository for other users, and dcommit with SVN is analogous to that.

When cloning an SVN repository, if none of the options for describing the repository layout is used (--trunk, --tags, --branches, --stdlayout), ’git svn clone’ will create a Git repository with completely linear history, where branches and tags appear as separate directories in the working copy. While this is the easiest way to get a copy of a complete repository, for projects with many branches it will lead to a working copy many times larger than just the trunk. Thus for projects using the standard directory structure (trunk/branches/tags), it is recommended to clone with option `--stdlayout`. If the project uses a non-standard structure, and/or if branches and tags are not required, it is easiest to only clone one directory (typically trunk), without giving any repository layout options. If the full history with branches and tags is required, the options `--trunk` / `--branches` / `--tags` must be used.

When using multiple --branches or --tags, ’git svn’ does not automatically handle name collisions (for example, if two branches from different paths have the same name, or if a branch and a tag have the same name). In these cases, use ’init’ to set up your Git repository then, before your first ’fetch’, edit the $GIT_DIR/config file so that the branches and tags are associated with different name spaces. For example:

```
branches = stable/*:refs/remotes/svn/stable/*
branches = debug/*:refs/remotes/svn/debug/*
```

CONFIGURATION ----------

’git svn’ stores [svn-remote] configuration information in the repository $GIT_DIR/config file. It is similar the core Git [remote] sections except ’fetch’ keys do not accept glob arguments; but they are instead handled by the ’branches’ and ’tags’ keys. Since some SVN repositories are oddly configured with multiple projects glob expansions such those listed below are allowed:

```
[svn-remote "project-a"]
    url = http://server.org/svn
    fetch = trunk/project-a:refs/remotes/project-a/trunk
    branches = branches/*/project-a:refs/remotes/project-a/branches/*
    branches = branches/release_*:refs/remotes/project-a/branches/release_*
    branches = branches/re*se:refs/remotes/project-a/branches/*
```

```
    tags = tags/*/project-a:refs/remotes/project-a/tags/*
```

Keep in mind that the `*` (asterisk) wildcard of the local ref (right of the `:`) **must** be the farthest right path component; however the remote wildcard may be anywhere as long as it's an independent path component (surrounded by `/` or EOL). This type of configuration is not automatically created by 'init' and should be manually entered with a text-editor or using 'git config'.

Also note that only one asterisk is allowed per word. For example:

```
branches = branches/re*se:refs/remotes/project-a/branches/*
```

will match branches 'release', 'rese', 're123se', however

```
branches = branches/re*s*e:refs/remotes/project-a/branches/*
```

will produce an error.

It is also possible to fetch a subset of branches or tags by using a comma-separated list of names within braces. For example:

```
[svn-remote "huge-project"]
    url = http://server.org/svn
    fetch = trunk/src:refs/remotes/trunk
    branches = branches/{red,green}/src:refs/remotes/project-a/branches/*
    tags = tags/{1.0,2.0}/src:refs/remotes/project-a/tags/*
```

Multiple fetch, branches, and tags keys are supported:

```
[svn-remote "messy-repo"]
    url = http://server.org/svn
    fetch = trunk/project-a:refs/remotes/project-a/trunk
    fetch = branches/demos/june-project-a-demo:refs/remotes/project-a/demos/june-demo
    branches = branches/server/*:refs/remotes/project-a/branches/*
    branches = branches/demos/2011/*:refs/remotes/project-a/2011-demos/*
    tags = tags/server/*:refs/remotes/project-a/tags/*
```

Creating a branch in such a configuration requires disambiguating which location to use using the -d or --destination flag:

```
$ git svn branch -d branches/server release-2-3-0
```

Note that git-svn keeps track of the highest revision in which a branch or tag has appeared. If the subset of branches or tags is changed after fetching, then $GIT_DIR/svn/.metadata must be manually edited to remove (or reset) branches-maxRev and/or tags-maxRev as appropriate.

FILES ----- $GIT_DIR/svn/***/.rev_map.**:: Mapping between Subversion revision numbers and Git commit names. In a repository where the noMetadata option is not set, this can be rebuilt from the git-svn-id: lines that are at the end of every commit (see the ’svn.noMetadata’ section above for details).
’git svn fetch’ and ’git svn rebase’ automatically update the rev_map if it is missing or not up to date. ’git svn reset’ automatically rewinds it.

BUGS ----

We ignore all SVN properties except svn:executable. Any unhandled properties are logged to $GIT_DIR/svn/<refname>/unhandled.log

Renamed and copied directories are not detected by Git and hence not tracked when committing to SVN. I do not plan on adding support for this as it’s quite difficult and time-consuming to get working for all the possible corner cases (Git doesn’t do it, either). Committing renamed and copied files is fully supported if they’re similar enough for Git to detect them.

In SVN, it is possible (though discouraged) to commit changes to a tag (because a tag is just a directory copy, thus technically the same as a branch). When cloning an SVN repository, ’git svn’ cannot know if such a commit to a tag will happen in the future. Thus it acts conservatively and imports all SVN tags as branches, prefixing the tag name with ’tags/’.

SEE ALSO -------- linkgit:git-rebase[1]

GIT --- Part of the linkgit:git[1] suite

□□□ #1
□ □□□ □□□ 2026-10-31 20:13:17 UTC
□ □□□ □□□ 2026-10-31 20:13:17 UTC