```
GIT-RESTORE(1)                    Git Manual                    GIT-RESTORE(1)

NAME
       git-restore - Restore working tree files

SYNOPSIS
       git restore [<options>] [--source=<tree>] [--staged] [--worktree] [--] <pathspec>...
       git restore [<options>] [--source=<tree>] [--staged] [--worktree] --pathspec-from-fi
le=<file> [--pathspec-file-nul]
       git restore (-p|--patch) [<options>] [--source=<tree>] [--staged] [--worktree] [--]
[<pathspec>...]

DESCRIPTION
       Restore specified paths in the working tree with some contents from a
       restore source. If a path is tracked but does not exist in the restore
       source, it will be removed to match the source.

       The command can also be used to restore the content in the index with
       --staged, or restore both the working tree and the index with --staged
       --worktree.

       By default, if --staged is given, the contents are restored from HEAD,
       otherwise from the index. Use --source to restore from a different
       commit.

       See "Reset, restore and revert" in git(1) for the differences between
       the three commands.

       THIS COMMAND IS EXPERIMENTAL. THE BEHAVIOR MAY CHANGE.

OPTIONS
       -s <tree>, --source=<tree>
           Restore the working tree files with the content from the given
           tree. It is common to specify the source tree by naming a commit,
           branch or tag associated with it.

           If not specified, the contents are restored from HEAD if --staged
           is given, otherwise from the index.

           As a special case, you may use "A...B" as a shortcut for the merge
           base of A and B if there is exactly one merge base. You can leave
           out at most one of A and B, in which case it defaults to HEAD.

       -p, --patch
           Interactively select hunks in the difference between the restore
           source and the restore location. See the "Interactive Mode" section
           of git-add(1) to learn how to operate the --patch mode.

           Note that --patch can accept no pathspec and will prompt to restore
           all modified paths.

       -W, --worktree, -S, --staged
           Specify the restore location. If neither option is specified, by
           default the working tree is restored. Specifying --staged will only
           restore the index. Specifying both restores both.

       -q, --quiet
           Quiet, suppress feedback messages. Implies --no-progress.

       --progress, --no-progress
           Progress status is reported on the standard error stream by default
           when it is attached to a terminal, unless --quiet is specified.
           This flag enables progress reporting even if not attached to a
           terminal, regardless of --quiet.

       --ours, --theirs
           When restoring files in the working tree from the index, use stage
           #2 (ours) or #3 (theirs) for unmerged paths.
           Note that during git rebase and git pull --rebase, ours and theirs
           may appear swapped. See the explanation of the same options in git-
           checkout(1) for details.

       -m, --merge
           When restoring files on the working tree from the index, recreate
           the conflicted merge in the unmerged paths.

       --conflict=<style>
           The same as --merge option above, but changes the way the
           conflicting hunks are presented, overriding the merge.conflictStyle
           configuration variable. Possible values are "merge" (default) and
           "diff3" (in addition to what is shown by "merge" style, shows the
           original contents).

       --ignore-unmerged
           When restoring files on the working tree from the index, do not
           abort the operation if there are unmerged entries and neither
           --ours, --theirs, --merge or --conflict is specified. Unmerged
           paths on the working tree are left alone.

       --ignore-skip-worktree-bits
           In sparse checkout mode, by default is to only update entries
           matched by <pathspec> and sparse patterns in
           $GIT_DIR/info/sparse-checkout. This option ignores the sparse
           patterns and unconditionally restores any files in <pathspec>.

       --recurse-submodules, --no-recurse-submodules
           If <pathspec> names an active submodule and the restore location
           includes the working tree, the submodule will only be updated if
           this option is given, in which case its working tree will be
           restored to the commit recorded in the superproject, and any local
           modifications overwritten. If nothing (or --no-recurse-submodules)
           is used, submodules working trees will not be updated. Just like
           git-checkout(1), this will detach HEAD of the submodule.

       --overlay, --no-overlay
           In overlay mode, the command never removes files when restoring. In
           no-overlay mode, tracked files that do not appear in the --source
           tree are removed, to make them match <tree> exactly. The default is
           no-overlay mode.

       --pathspec-from-file=<file>
           Pathspec is passed in <file> instead of commandline args. If <file>
           is exactly - then standard input is used. Pathspec elements are
           separated by LF or CR/LF. Pathspec elements can be quoted as
           explained for the configuration variable core.quotePath (see git-
           config(1)). See also --pathspec-file-nul and global
           --literal-pathspecs.

       --pathspec-file-nul
           Only meaningful with --pathspec-from-file. Pathspec elements are
           separated with NUL character and all other characters are taken
           literally (including newlines and quotes).

       --
           Do not interpret any more arguments as options.

       <pathspec>...
           Limits the paths affected by the operation.

           For more details, see the pathspec entry in gitglossary(7).

EXAMPLES
       The following sequence switches to the master branch, reverts the
       Makefile to two revisions back, deletes hello.c by mistake, and gets it
       back from the index.
```

```
    $ git switch master
    $ git restore --source master~2 Makefile  (1)
    $ rm -f hello.c
    $ git restore hello.c                     (2)
```

1. take a file out of another commit
2. restore hello.c from the index

If you want to restore all C source files to match the version in the index, you can say

```
    $ git restore '*.c'
```

Note the quotes around *.c. The file hello.c will also be restored, even though it is no longer in the working tree, because the file globbing is used to match entries in the index (not in the working tree by the shell).

To restore all files in the current directory

```
    $ git restore .
```

or to restore all working tree files with top pathspec magic (see gitglossary(7))

```
    $ git restore :/
```

To restore a file in the index to match the version in HEAD (this is the same as using git-reset(1))

```
    $ git restore --staged hello.c
```

or you can restore both the index and the working tree (this the same as using git-checkout(1))

```
    $ git restore --source=HEAD --staged --worktree hello.c
```

or the short form which is more practical but less readable:

```
    $ git restore -s@ -SW hello.c
```

SEE ALSO

git-checkout(1), git-reset(1)

GIT

Part of the git(1) suite