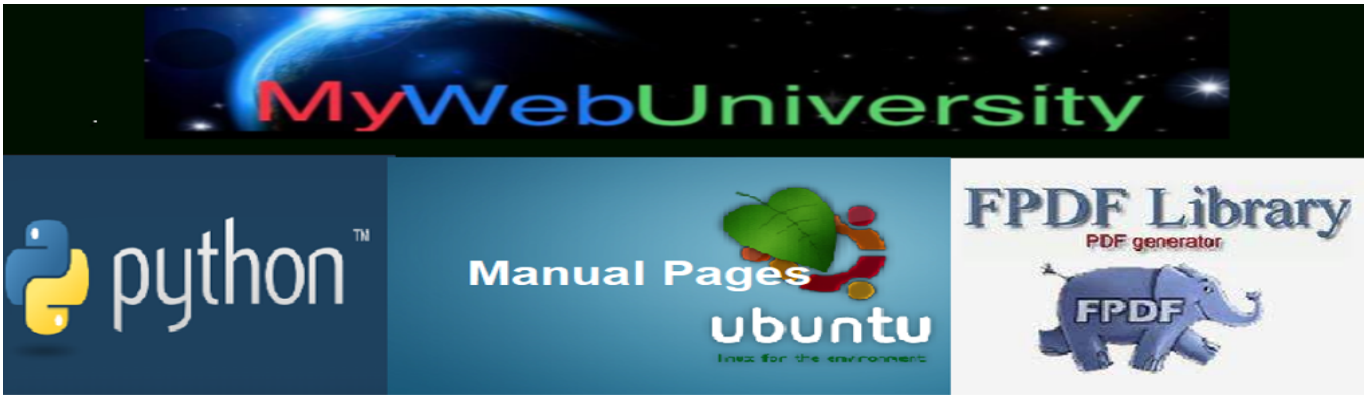

Rocky Enterprise Linux 9.2 Manual Pages on command 'git-sparse-checkout.1'

$ man git-sparse-checkout.1

GIT-SPARSE-CHECKOU(1) Git Manual GIT-SPARSE-CHECKOU(1)

NAME

git-sparse-checkout - Initialize and modify the sparse-checkout configuration, which reduces the checkout to a set of paths given by a list of patterns.

SYNOPSIS

git sparse-checkout <subcommand> [options]

DESCRIPTION

Initialize and modify the sparse-checkout configuration, which reduces the checkout to a set of paths given by a list of patterns.

THIS COMMAND IS EXPERIMENTAL. ITS BEHAVIOR, AND THE BEHAVIOR OF OTHER COMMANDS IN THE PRESENCE OF SPARSE-CHECKOUTS, WILL LIKELY CHANGE IN THE FUTURE.

COMMANDS

list

Describe the patterns in the sparse-checkout file.

init

Enable the core.sparseCheckout setting. If the sparse-checkout file does not exist, then populate it with patterns that match every file in the root directory and no other directories, then will remove all directories tracked by Git. Add patterns to the sparse-checkout file to repopulate the working directory.

To avoid interfering with other worktrees, it first enables the extensions.worktreeConfig setting and makes sure to set the core.sparseCheckout setting in the worktree-specific config file.

When --cone is provided, the core.sparseCheckoutCone setting is also set, allowing for

better performance with a limited set of patterns (see CONE PATTERN SET below).

Use the --[no-]sparse-index option to toggle the use of the sparse index format. This reduces the size of the index to be more closely aligned with your sparse-checkout definition. This can have significant performance advantages for commands such as git status or git add. This feature is still experimental. Some commands might be slower with a sparse index until they are properly integrated with the feature.

WARNING: Using a sparse index requires modifying the index in a way that is not completely understood by external tools. If you have trouble with this compatibility, then run git sparse-checkout init --no-sparse-index to rewrite your index to not be sparse. Older versions of Git will not understand the sparse directory entries index extension and may fail to interact with your repository until it is disabled.

set

Write a set of patterns to the sparse-checkout file, as given as a list of arguments following the set subcommand. Update the working directory to match the new patterns.

Enable the core.sparseCheckout config setting if it is not already enabled.

When the --stdin option is provided, the patterns are read from standard in as a newline-delimited list instead of from the arguments.

When core.sparseCheckoutCone is enabled, the input list is considered a list of directories instead of sparse-checkout patterns. The command writes patterns to the sparse-checkout file to include all files contained in those directories (recursively) as well as files that are siblings of ancestor directories. The input format matches the output of git ls-tree --name-only. This includes interpreting pathnames that begin with a double quote (") as C-style quoted strings.

add

Update the sparse-checkout file to include additional patterns. By default, these patterns are read from the command-line arguments, but they can be read from stdin using the --stdin option. When core.sparseCheckoutCone is enabled, the given patterns are interpreted as directory names as in the set subcommand.

reapply

Reapply the sparsity pattern rules to paths in the working tree. Commands like merge or rebase can materialize paths to do their work (e.g. in order to show you a conflict), and other sparse-checkout commands might fail to sparsify an individual file (e.g. because it has unstaged changes or conflicts). In such cases, it can make

sense to run git sparse-checkout reapply later after cleaning up affected paths (e.g. resolving conflicts, undoing or committing changes, etc.).

disable

Disable the core.sparseCheckout config setting, and restore the working directory to include all files. Leaves the sparse-checkout file intact so a later git sparse-checkout init command may return the working directory to the same state.

SPARSE CHECKOUT

"Sparse checkout" allows populating the working directory sparsely. It uses the skip-worktree bit (see git-update-index(1)) to tell Git whether a file in the working directory is worth looking at. If the skip-worktree bit is set, then the file is ignored in the working directory. Git will not populate the contents of those files, which makes a sparse checkout helpful when working in a repository with many files, but only a few are important to the current user.

The $GIT_DIR/info/sparse-checkout file is used to define the skip-worktree reference bitmap. When Git updates the working directory, it updates the skip-worktree bits in the index based on this file. The files matching the patterns in the file will appear in the working directory, and the rest will not.

To enable the sparse-checkout feature, run git sparse-checkout init to initialize a simple sparse-checkout file and enable the core.sparseCheckout config setting. Then, run git sparse-checkout set to modify the patterns in the sparse-checkout file.

To repopulate the working directory with all files, use the git sparse-checkout disable command.

FULL PATTERN SET

By default, the sparse-checkout file uses the same syntax as .gitignore files.

While $GIT_DIR/info/sparse-checkout is usually used to specify what files are included, you can also specify what files are not included, using negative patterns. For example, to remove the file unwanted:

```
/*
!unwanted
```

CONE PATTERN SET

The full pattern set allows for arbitrary pattern matches and complicated inclusion/exclusion rules. These can result in O(N*M) pattern matches when updating the index, where N is the number of patterns and M is the number of paths in the index. To

combat this performance issue, a more restricted pattern set is allowed when core.sparseCheckoutCone is enabled.

The accepted patterns in the cone pattern set are:

1. Recursive: All paths inside a directory are included.
2. Parent: All files immediately inside a directory are included.

In addition to the above two patterns, we also expect that all files in the root directory are included. If a recursive pattern is added, then all leading directories are added as parent patterns.

By default, when running git sparse-checkout init, the root directory is added as a parent pattern. At this point, the sparse-checkout file contains the following patterns:

```
/*
!/*/
```

This says "include everything in root, but nothing two levels below root."

When in cone mode, the git sparse-checkout set subcommand takes a list of directories instead of a list of sparse-checkout patterns. In this mode, the command git sparse-checkout set A/B/C sets the directory A/B/C as a recursive pattern, the directories A and A/B are added as parent patterns. The resulting sparse-checkout file is now

```
/*
!/*/
/A/
!/A/*/
/A/B/
!/A/B/*/
/A/B/C/
```

Here, order matters, so the negative patterns are overridden by the positive patterns that appear lower in the file.

If core.sparseCheckoutCone=true, then Git will parse the sparse-checkout file expecting patterns of these types. Git will warn if the patterns do not match. If the patterns do match the expected format, then Git will use faster hash- based algorithms to compute inclusion in the sparse-checkout.

In the cone mode case, the git sparse-checkout list subcommand will list the directories that define the recursive patterns. For the example sparse-checkout file above, the output is as follows:

```
$ git sparse-checkout list
A/B/C
```

If core.ignoreCase=true, then the pattern-matching algorithm will use a case-insensitive check. This corrects for case mismatched filenames in the git sparse-checkout set command to reflect the expected cone in the working directory.

When changing the sparse-checkout patterns in cone mode, Git will inspect each tracked directory that is not within the sparse-checkout cone to see if it contains any untracked files. If all of those files are ignored due to the .gitignore patterns, then the directory will be deleted. If any of the untracked files within that directory is not ignored, then no deletions will occur within that directory and a warning message will appear. If these files are important, then reset your sparse-checkout definition so they are included, use git add and git commit to store them, then remove any remaining files manually to ensure Git can behave optimally.

SUBMODULES

If your repository contains one or more submodules, then submodules are populated based on interactions with the git submodule command. Specifically, git submodule init -- <path> will ensure the submodule at <path> is present, while git submodule deinit [-f] -- <path> will remove the files for the submodule at <path> (including any untracked files, uncommitted changes, and unpushed history). Similar to how sparse-checkout removes files from the working tree but still leaves entries in the index, deinitialized submodules are removed from the working directory but still have an entry in the index.

Since submodules may have unpushed changes or untracked files, removing them could result in data loss. Thus, changing sparse inclusion/exclusion rules will not cause an already checked out submodule to be removed from the working copy. Said another way, just as checkout will not cause submodules to be automatically removed or initialized even when switching between branches that remove or add submodules, using sparse-checkout to reduce or expand the scope of "interesting" files will not cause submodules to be automatically deinitialized or initialized either.

Further, the above facts mean that there are multiple reasons that "tracked" files might not be present in the working copy: sparsity pattern application from sparse-checkout, and submodule initialization state. Thus, commands like git grep that work on tracked files in the working copy may return results that are limited by either or both of these restrictions.

SEE ALSO

git-read-tree(1) gitignore(5)

GIT

Part of the git(1) suite