

**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.

*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.

*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." If we then add the folder **A/B/C** as a recursive pattern, the folders **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.

## SUBMODULES

If your repository contains one or more submodules, then those submodules will appear based on which you initialized with the **git submodule** command. If your sparse-checkout patterns exclude an initialized submodule, then that submodule will still appear in your working directory.

## SEE ALSO

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

## GIT

Part of the **git(1)** suite