NAME

git-sparse-checkout - Reduce your working tree to a subset of tracked files

SYNOPSIS

git sparse-checkout (init | list | set | add | reapply | disable | check-rules) [<options>]

DESCRIPTION

This command is used to create sparse checkouts, which change the working tree from having all tracked files present to only having a subset of those files. It can also switch which subset of files are present, or undo and go back to having all tracked files present in the working copy.

The subset of files is chosen by providing a list of directories in cone mode (the default), or by providing a list of patterns in non-cone mode.

When in a sparse-checkout, other Git commands behave a bit differently. For example, switching branches will not update paths outside the sparse-checkout directories/patterns, and **git commit -a** will not record paths outside the sparse-checkout directories/patterns as deleted.

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 directories or patterns in the sparse-checkout file.

set

Enable the necessary sparse-checkout config settings (**core.sparseCheckout**, **core.sparseCheckoutCone**, and **index.sparse**) if they are not already set to the desired values, populate the sparse-checkout file from the list of arguments following the *set* subcommand, and update the working directory to match.

To ensure that adjusting the sparse-checkout settings within a worktree does not alter the sparse-checkout settings in other worktrees, the *set* subcommand will upgrade your repository config to use worktree-specific config if not already present. The sparsity defined by the arguments to the *set* subcommand are stored in the worktree-specific sparse-checkout file. See **git-worktree**(1) and the documentation of **extensions.worktreeConfig** in **git-config**(1) for more details.

When the **--stdin** option is provided, the directories or patterns are read from standard in as a

newline-delimited list instead of from the arguments.

By default, the input list is considered a list of directories, matching the output of **git ls-tree -d --name-only**. This includes interpreting pathnames that begin with a double quote (") as C-style quoted strings. Note that all files under the specified directories (at any depth) will be included in the sparse checkout, as well as files that are siblings of either the given directory or any of its ancestors (see *CONE PATTERN SET* below for more details). In the past, this was not the default, and **--cone** needed to be specified or **core.sparseCheckoutCone** needed to be enabled.

When **--no-cone** is passed, the input list is considered a list of patterns. This mode has a number of drawbacks, including not working with some options like **--sparse-index**. As explained in the "Non-cone Problems" section below, we do not recommend using it.

Use the **--[no-]sparse-index** option to use a sparse index (the default is to not use it). A sparse index 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.

add

Update the sparse-checkout file to include additional directories (in cone mode) or patterns (in non-cone mode). By default, these directories or patterns are read from the command-line arguments, but they can be read from stdin using the **--stdin** option.

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.).

The **reapply** command can also take **--[no-]cone** and **--[no-]sparse-index** flags, with the same meaning as the flags from the **set** command, in order to change which sparsity mode you are using without needing to also respecify all sparsity paths.

disable

Disable the **core.sparseCheckout** config setting, and restore the working directory to include all files.

init

Deprecated command that behaves like **set** with no specified paths. May be removed in the future.

Historically, **set** did not handle all the necessary config settings, which meant that both **init** and **set** had to be called. Invoking both meant the **init** step would first remove nearly all tracked files (and in cone mode, ignored files too), then the **set** step would add many of the tracked files (but not ignored files) back. In addition to the lost files, the performance and UI of this combination was poor.

Also, historically, **init** would not actually initialize the sparse-checkout file if it already existed. This meant it was possible to return to a sparse-checkout without remembering which paths to pass to a subsequent *set* or *add* command. However, **--cone** and **--sparse-index** options would not be remembered across the disable command, so the easy restore of calling a plain **init** decreased in utility.

check-rules

Check whether sparsity rules match one or more paths.

By default **check-rules** reads a list of paths from stdin and outputs only the ones that match the current sparsity rules. The input is expected to consist of one path per line, matching the output of **git ls-tree --name-only** including that pathnames that begin with a double quote (") are interpreted as C-style quoted strings.

When called with the **--rules-file <file>** flag the input files are matched against the sparse checkout rules found in **<file>** instead of the current ones. The rules in the files are expected to be in the same form as accepted by **git sparse-checkout set --stdin** (in particular, they must be newline-delimited).

By default, the rules passed to the **--rules-file** option are interpreted as cone mode directories. To pass non-cone mode patterns with **--rules-file**, combine the option with the **--no-cone** option.

When called with the **-z** flag, the format of the paths input on stdin as well as the output paths are \0 terminated and not quoted. Note that this does not apply to the format of the rules passed with the **--rules-file** option.

EXAMPLES

git sparse-checkout set MY/DIR1 SUB/DIR2

Change to a sparse checkout with all files (at any depth) under MY/DIR1/ and SUB/DIR2/ present in the working copy (plus all files immediately under MY/ and SUB/ and the toplevel directory). If already in a sparse checkout, change which files are present in the working copy to this new selection. Note that this command will also delete all ignored files in any directory that no longer has either tracked or non-ignored-untracked files present.

git sparse-checkout disable

Repopulate the working directory with all files, disabling sparse checkouts.

git sparse-checkout add SOME/DIR/ECTORY

Add all files under SOME/DIR/ECTORY/ (at any depth) to the sparse checkout, as well as all files immediately under SOME/DIR/ and immediately under SOME/. Must already be in a sparse checkout before using this command.

git sparse-checkout reapply

It is possible for commands to update the working tree in a way that does not respect the selected sparsity directories. This can come from tools external to Git writing files, or even affect Git commands because of either special cases (such as hitting conflicts when merging/rebasing), or because some commands didn't fully support sparse checkouts (e.g. the old **recursive** merge backend had only limited support). This command reapplies the existing sparse directory specifications to make the working directory match.

INTERNALS -- 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, and the file is not present in the working tree, then its absence is ignored. Git will avoid populating 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.

INTERNALS -- NON-CONE PROBLEMS

The **$GIT_DIR/info/sparse-checkout** file populated by the **set** and **add** subcommands is defined to be a bunch of patterns (one per line) using the same syntax as **.gitignore** files. In cone mode, these patterns are restricted to matching directories (and users only ever need supply or see directory names), while in non-cone mode any gitignore-style pattern is permitted. Using the full gitignore-style patterns in non-cone mode has a number of shortcomings:

- it makes various worktree-updating processes (pull, merge, rebase, switch, reset, checkout, etc.) require O(N*M) pattern matches, where N is the number of patterns and M is the number of paths in the index. This scales poorly.

- the scaling issue has to be done via limiting the number of patterns via specifying leading directory name or glob.

- globs on the command line is error-prone as users may forget to quote the glob, causing the shell to expand it into all matching files and pass them all individually along to sparse-checkout set/add. While this could also be a problem with e.g. "git grep -- *.c", mistakes with grep/log/status appear in the immediate output. With sparse-checkout, the mistake gets recorded at the time the sparse-checkout command is run and might not be problematic until the user later switches branches or rebases or merges, thus putting a delay between the user's error and when they have a chance to catch/notice it.

- to the previous item, sparse-checkout has an *add* subcommand but no *remove* subcommand. Even if a *remove* subcommand were added, undoing an accidental unquoted glob runs the risk of "removing too much", as it may remove entries that had been included before the accidental add.

- mode uses gitignore-style patterns to select what to **include** (with the exception of negated patterns), while .gitignore files use gitignore-style patterns to select what to **exclude** (with the exception of negated patterns). The documentation on gitignore-style patterns usually does not talk in terms of matching or non-matching, but on what the user wants to "exclude". This can cause confusion for users trying to learn how to specify sparse-checkout patterns to get their desired behavior.

- other git subcommand that wants to provide "special path pattern matching" of some sort uses pathspecs, but non-cone mode for sparse-checkout uses gitignore patterns, which feels inconsistent.

- has edge cases where the "right" behavior is unclear. Two examples:

  First, two users are in a subdirectory, and the first runs
  ```
  git sparse-checkout set '/toplevel-dir/*.c'
  ```
  while the second runs
  ```
  git sparse-checkout set relative-dir
  ```

Should those arguments be transliterated into
current/subdirectory/toplevel-dir/*.c
and
current/subdirectory/relative-dir
before inserting into the sparse-checkout file? The use
the first command is probably aware that arguments to
supposed to be patterns in non-cone mode, and probabl
happy with such a transliteration. However, many gitig
patterns are just paths, which might be what the user w
second command was thinking, and they'd be upset if t
wasn't transliterated.

Second, what should bash-completion complete on for set/add commands for non-cone users? If it suggests paths, is it exacerbating the problem above? Also, if it suggests paths, what if the user has a file or directory that begins with either a '!' or '#' or has a '*', '\', '?', '[', or ']' in its name? And if it suggests paths, will it complete "/pro" to "/proc" (in the root filesystem) rather than to "/progress.txt" in the current directory? (Note that users are likely to want to start paths with a leading '/' in non-cone mode, for the same reason that .gitignore files often have one.) Completing on files or directories might give nasty surprises in all these cases.

- excessive flexibility made other extensions essentially impractical. **--sparse-index** is likely impossible in non-cone mode; even if it is somehow feasible, it would have been far more work to implement and may have been too slow in practice. Some ideas for adding coupling between partial clones and sparse checkouts are only practical with a more restricted set of paths as well.

For all these reasons, non-cone mode is deprecated. Please switch to using cone mode.

INTERNALS -- CONE MODE HANDLING

The "cone mode", which is the default, lets you specify only what directories to include. For any directory specified, all paths below that directory will be included, and any paths immediately under leading directories (including the toplevel directory) will also be included. Thus, if you specified the directory Documentation/technical/ then your sparse checkout would contain:

- files in the toplevel-directory

- files immediately under Documentation/

- files at any depth under Documentation/technical/

Also, in cone mode, even if no directories are specified, then the files in the toplevel directory will be included.

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.

See also the "Internals -- Cone Pattern Set" section to learn how the directories are transformed under the hood into a subset of the Full Pattern Set of sparse-checkout.

INTERNALS -- FULL 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 sparse-checkout file uses the same syntax as **.gitignore** files; see **gitignore**(5) for details. Here, though, the patterns are usually being used to select which files to include rather than which files to exclude. (However, it can get a bit confusing since gitignore-style patterns have negations defined by patterns which begin with a *!*, so you can also select files to *not* include.)

For example, to select everything, and then to remove the file **unwanted** (so that every file will appear in your working tree except the file named **unwanted**):

```
git sparse-checkout set --no-cone '/*' '!unwanted'
```

These patterns are just placed into the **$GIT_DIR/info/sparse-checkout** as-is, so the contents of that file at this point would be

```
/*
```

```
!unwanted
```

See also the "Sparse Checkout" section of **git-read-tree**(1) to learn more about the gitignore-style patterns used in sparse checkouts.

INTERNALS -- CONE PATTERN SET

In cone mode, only directories are accepted, but they are translated into the same gitignore-style patterns used in the full pattern set. We refer to the particular patterns used in those mode as being of one of two types:

1. All paths inside a directory are included.

2. All files immediately inside a directory are included.

Since cone mode always includes files at the toplevel, when running **git sparse-checkout set** with no directories specified, the toplevel directory is added as a parent pattern. At this point, the sparse-checkout file contains the following patterns:

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

This says "include everything immediately under the toplevel directory, but nothing at any level below that."

When in cone mode, the **git sparse-checkout set** subcommand takes a list of directories. 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.

Unless **core.sparseCheckoutCone** is explicitly set to **false**, 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. If they do not match, git will behave as though **core.sparseCheckoutCone** was false, regardless of its setting.

In the cone mode case, despite the fact that full patterns are written to the $GIT_DIR/info/sparse-checkout file, 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.

INTERNALS -- 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