KIP Intern Git Assignment
Topic: -add
Group 12
Name: Muskan Jain(1736)

NAME

git-add - Add file contents to the index

SYNOPSIS

```
git add [--verbose | -v] [--dry-run | -n] [--force | -f] [--interactive |
-i] [--patch | -p]
          [--edit | -e] [--[no-]all | --[no-]ignore-removal | [--update |
-u]] [--sparse]
          [--intent-to-add | -N] [--refresh] [--ignore-errors] [--ignore-
missing] [--renormalize]
          [--chmod=(+|-)x] [--pathspec-from-file=<file> [--pathspec-file-
nul]]
          [--] [<pathspec>…]
```

DESCRIPTION

This command updates the index using the current content found in the working tree, to prepare the content staged for the next commit. It typically adds the current content of existing paths as a whole, but with some options it can also be used to add content with only part of the changes made to the working tree files applied, or remove paths that do not exist in the working tree anymore.

The "index" holds a snapshot of the content of the working tree, and it is this snapshot that is taken as the contents of the next commit. Thus after making any changes to the working tree, and before running the commit command, you must use the `add` command to add any new or modified files to the index.

This command can be performed multiple times before a commit. It only adds the content of the specified file(s) at the time the add command is run; if you want subsequent changes included in the next commit, then you must run `git add` again to add the new content to the index.

The `git status` command can be used to obtain a summary of which files have changes that are staged for the next commit.

The `git add` command will not add ignored files by default. If any ignored files were explicitly specified on the command line, `git add` will fail with a list of ignored files. Ignored files reached by directory recursion or filename globbing performed by Git (quote your globs before the shell) will be silently ignored. The git add command can be used to add ignored files with the `-f` (force) option.

Please see git-commit[1] for alternative ways to add content to a commit.

OPTIONS

--renormalize

Apply the "clean" process freshly to all tracked files to forcibly add them again to the index. This is useful after changing core.autocrlf configuration or the text attribute in order to correct files added with wrong CRLF/LF line endings. This option implies -u.

--chmod=(+|-)x

Override the executable bit of the added files. The executable bit is only changed in the index, the files on disk are left unchanged.

--pathspec-from-file=<file>

Pathspec is passed in <file> instead of command line 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).

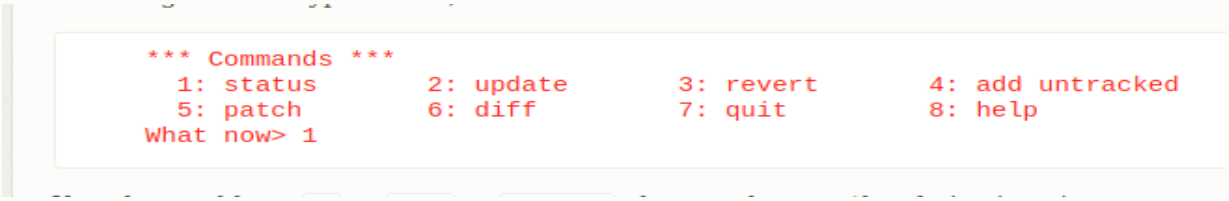
--

This option can be used to separate command-line options from the list of files, (useful when filenames might be mistaken for command-line options).

INTERACTIVE MODE

When the command enters the interactive mode, it shows the output of the **status** subcommand, and then goes into its interactive command loop.

The command loop shows the list of subcommands available, and gives a prompt "What now> ". In general, when the prompt ends with a single >, you can pick only one of the choices given and type return, like this:



You also could say `s` or `sta` or `status` above as long as the choice is unique.

The main command loop has 6 subcommands (plus help and quit).

status

This shows the change between HEAD and index (i.e. what will be committed if you say `git commit`), and between index and working tree files (i.e. what you could stage further before `git commit` using `git add`) for each path. A sample output looks like this:

```
           staged     unstaged path
 1:       binary      nothing foo.png
 2:     +403/-35        +1/-1 git-add--interactive.perl
```

It shows that foo.png has differences from HEAD (but that is binary so line count cannot be shown) and there is no difference between indexed copy and the working tree version (if the working tree version were also different, **binary** would have been shown in place of **nothing**). The other file, git-add--interactive.perl, has 403 lines added and 35 lines deleted if you commit what is in the index, but working tree file has further modifications (one addition and one deletion).

update

This shows the status information and issues an "Update>>" prompt. When the prompt ends with double >>, you can make more than one selection, concatenated with whitespace or comma. Also you can say ranges. E.g. "2-5 7,9" to choose 2,3,4,5,7,9 from the list. If the second number in a range is omitted, all remaining patches are taken. E.g. "7-" to choose 7,8,9 from the list. You can say * to choose everything.

What you chose are then highlighted with *, like this:

```
           staged     unstaged path
  1:       binary      nothing foo.png
* 2:     +403/-35        +1/-1 git-add--interactive.perl
```

revert

This has a very similar UI to **update**, and the staged information for selected paths are reverted to that of the HEAD version. Reverting new paths makes them untracked.

add untracked

This has a very similar UI to **update** and **revert**, and lets you add untracked paths to the index.

patch

This lets you choose one path out of a **status** like selection. After choosing the path, it presents the diff between the index and the working tree file and asks you if you want to stage the change of each hunk. You can select one of the following options and type return:

```
y - stage this hunk
n - do not stage this hunk
q - quit; do not stage this hunk or any of the remaining ones
a - stage this hunk and all later hunks in the file
d - do not stage this hunk or any of the later hunks in the file
g - select a hunk to go to
/ - search for a hunk matching the given regex
j - leave this hunk undecided, see next undecided hunk
J - leave this hunk undecided, see next hunk
k - leave this hunk undecided, see previous undecided hunk
K - leave this hunk undecided, see previous hunk
s - split the current hunk into smaller hunks
e - manually edit the current hunk
? - print help
```

After deciding the fate for all hunks, if there is any hunk that was chosen, the index is updated with the selected hunks.

You can omit having to type return here, by setting the configuration variable `interactive.singleKey` to `true`.

diff

This lets you review what will be committed (i.e. between HEAD and index).

EDITING PATCHES

Invoking `git add -e` or selecting `e` from the interactive hunk selector will open a patch in your editor; after the editor exits, the result is applied to the index. You are free to make arbitrary changes to the patch, but note that some changes may have confusing results, or even result in a patch that cannot be applied. If you want to abort the operation entirely (i.e., stage nothing new in the index), simply delete all lines of the patch. The list below describes some common things you may see in a patch, and which editing operations make sense on them.

modifying existing content

One can also modify context lines by staging them for removal (by converting " " to "-") and adding a "+" line with the new content. Similarly, one can modify "+" lines for existing additions or modifications. In all cases, the new modification will appear reverted in the working tree.

new content

You may also add new content that does not exist in the patch; simply add new lines, each starting with "+". The addition will appear reverted in the working tree.

There are also several operations which should be avoided entirely, as they will make the patch impossible to apply:

- adding context (" ") or removal ("-") lines
- deleting context or removal lines
- modifying the contents of context or removal lines