Help Topic: The Git Status Command

Understanding Your Tree Status and useful Aliases

Spring 2022

Oscar Viquez, oviquezr@mit.edu
Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139

The Git `status` Command

The `git status` command informs you about what has changed in your local repository checkout compared to the moment you first checked it out or last did an update via `git pull`. It does not compare it to the contents of the Git server. For this reason `git status` may be invoked offline. This is a tool I find valuable for confirming what exactly I may have done to my local checkout. This is important right before committing changes, to ensure that the desired changes are staged for the next commit. More info on `git status` can always be found by Googling "Git book" and reading the full PDF online free, or just typing `git help status` anytime on the command line.

Basic usage of the `git status` command

The basic syntax for the `git status` command is:

```
$ git status       # (invoked somewhere within previously checked out tree)
```

The `git status` does *not* require a network connection to the server and only indicates changes compared to your initial checkout or previous update.

Deciphering the output of the `git status` command

The `git status` output may look something like the below by way of example:

```
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
  modified:   app1/app1.cpp
  new file:   app1/main.cpp

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
  modified:   app1/app1.h

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  app2/app2.cpp
  app2/app2.h
```

Files that have not been changed, deleted, or added or are any way different compared to the last update are *not* reported. Otherwise each line reports the file name, grouped by how it would affect the repository. In the example above, the output tells us:

- The file `app1/app1.cpp` is already known to Git, and has been modified locally compared to the last update. The changes are staged for the next commit, where they will be written to the repository's history.
- The file `app1/main.cpp` is new to Git, has been newly added by you, and will be written to the repository on the next commit.
- The file `app1/app1.h` is already known to Git, but has been modified locally compared to the last update. The changes are not currently staged for commit.
- The files `app2/app2.h` and `app2/app2.cpp` are in your local file system but they are completely unknown to Git. They were not part of the checkout or last update, and no action will be taken on them during the next commit. Possibly they should be rightfully ignored, or perhaps you have forgot to `git add` them.

Generally you will encounter files having one of the above modes. There are others however, and you can read about them in the git documentation, or get a glimpse of them with `git help status`.

Using the `git status` command in quiet mode

The `git status` command may be used with the optional argument, `--untracked-files[=<mode>]`, or simply `-u[<mode>]`. This can be used to manage the level of detail for unversioned items. The possible modes are:

- no: skip all untracked files and folders.
- normal: show all untracked files, and untracked directories.
- all: also show individual files contained within untracked directories.

If no mode is set, the default is `all`. In the example above, the `-uno` option would produce the following output instead:

```
$ git status -uno
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
  modified:   app1/app1.cpp
  new file:   app1/main.cpp

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
  modified:   app1/app1.h
```

The last two files, `app2/app2.h` and `app2/app2.cpp` are suppressed. The `-uno` option is very useful when your local repository has many automatically generated files that are not already caught by the filters in the .gitignore files in the repository. This could include object files if you're working in C++, or other temporary files if you're working with LaTeX, for example. In these cases the number of such lines can overwhelm the output of the `git status` command, rendering it hard to use.