Pro Git(https://git-scm.com/book/ko/v2/)를 바탕으로 작성하였습니다.

YoonJoon Lee
SoC KAIST

Fall 2018

What we will take a look in this series

1. Getting Started
2. Git Basics
3. Git Branch
4. Git Server - GitLab

What we will take a look today

1. Getting Started
2. Git Basics
3. Git Branch
4. Git Server - GitLab

What I will talk about in this section

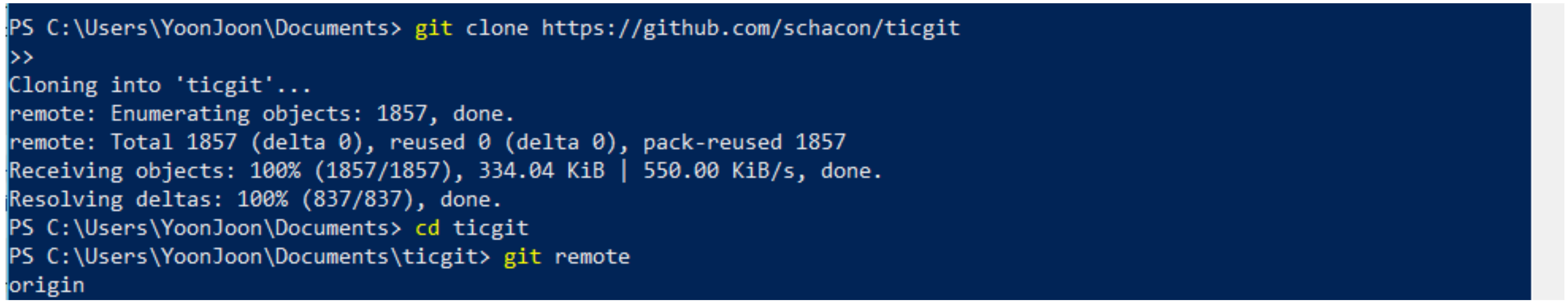
1. Getting a Git Repository
2. Recording Changes to the Repository
3. Viewing the Commit History
4. Undoing Things
5. Working with Remotes
6. Tagging
7. Git Aliases

Working with Remotes

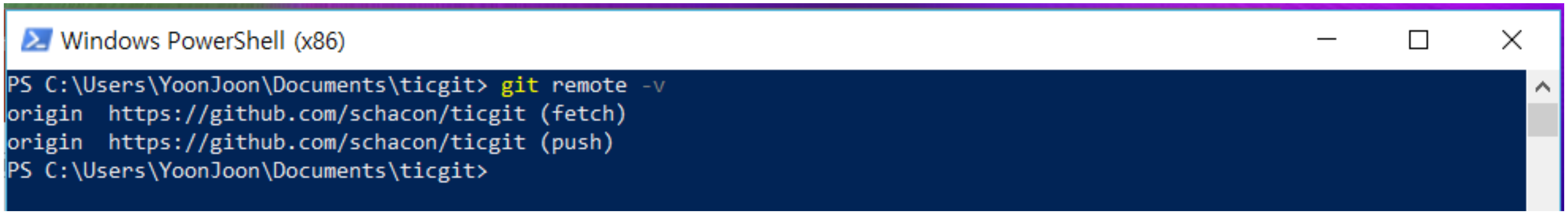
- To be able to collaborate on any Git project, we need to know how to manage our remote repositories.
- Remote repositories are versions of our project that are hosted on the Internet or network somewhere.

Showing Remotes

To see which remote servers we have configured, we can run the `git remote` command.

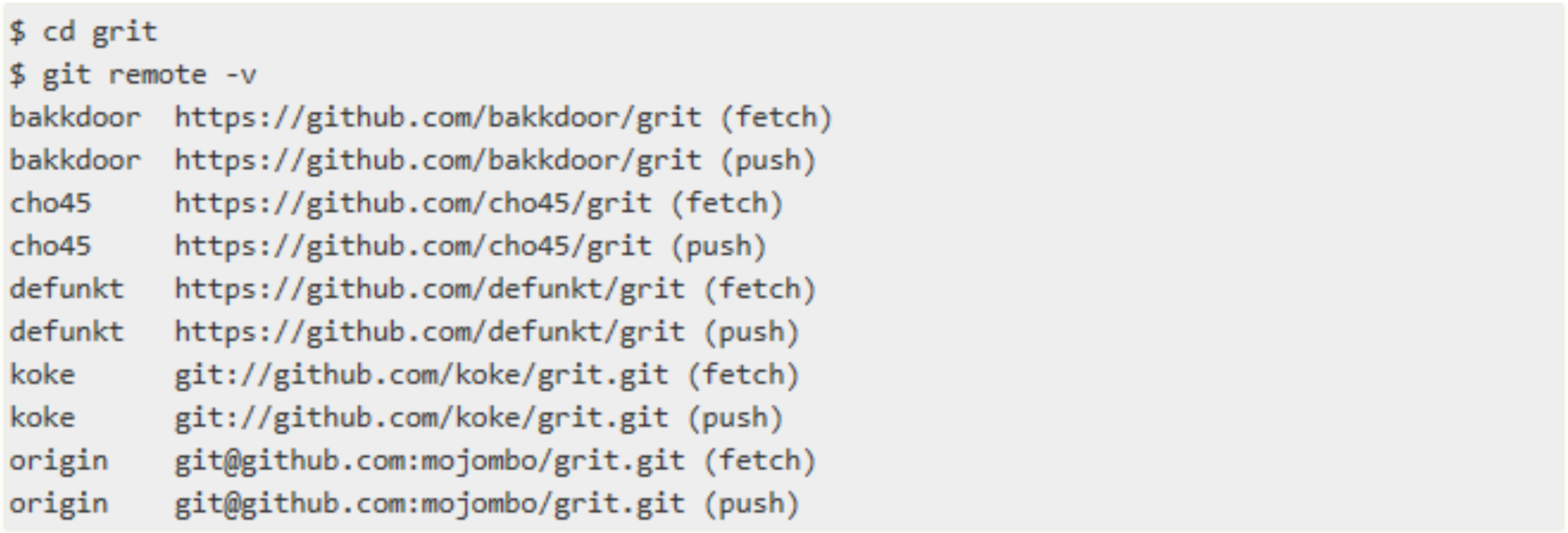

```
PS C:\Users\YoonJoon\Documents> git clone https://github.com/schacon/ticgit
>>
Cloning into 'ticgit'...
remote: Enumerating objects: 1857, done.
remote: Total 1857 (delta 0), reused 0 (delta 0), pack-reused 1857
Receiving objects: 100% (1857/1857), 334.04 KiB | 550.00 KiB/s, done.
Resolving deltas: 100% (837/837), done.
PS C:\Users\YoonJoon\Documents> cd ticgit
PS C:\Users\YoonJoon\Documents\ticgit> git remote
origin
```

We can also specify `-v`, which shows us the URLs that Git has stored for the shortname to be used when reading and writing to that remote:



```
PS C:\Users\YoonJoon\Documents\ticgit> git remote -v
origin  https://github.com/schacon/ticgit (fetch)
origin  https://github.com/schacon/ticgit (push)
PS C:\Users\YoonJoon\Documents\ticgit>
```

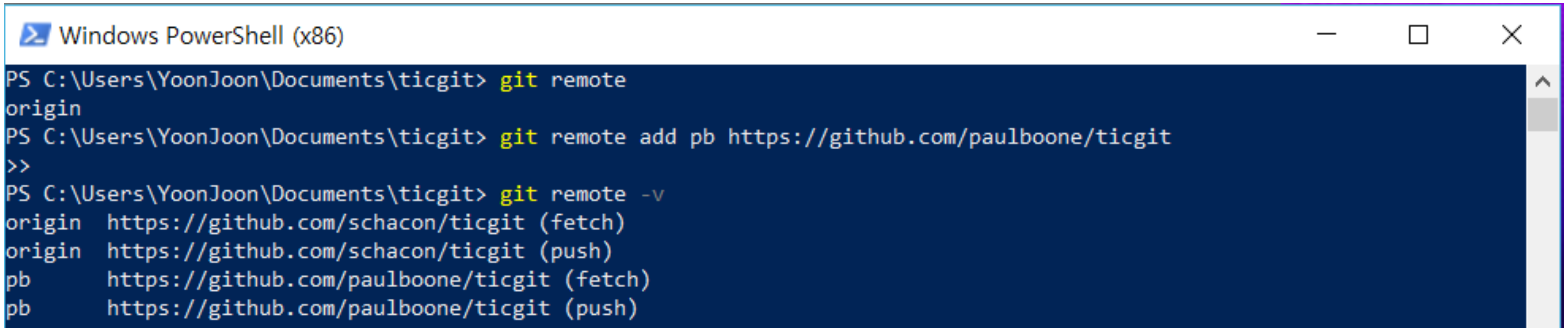
If we have more than one remote, the command lists them all. For example, a repository with multiple remotes for working with several collaborators might look something like this.



```
$ cd grit
$ git remote -v
bakkdoor  https://github.com/bakkdoor/grit (fetch)
bakkdoor  https://github.com/bakkdoor/grit (push)
cho45     https://github.com/cho45/grit (fetch)
cho45     https://github.com/cho45/grit (push)
defunkt   https://github.com/defunkt/grit (fetch)
defunkt   https://github.com/defunkt/grit (push)
koke      git://github.com/koke/grit.git (fetch)
koke      git://github.com/koke/grit.git (push)
origin    git@github.com:mojombo/grit.git (fetch)
origin    git@github.com:mojombo/grit.git (push)
```

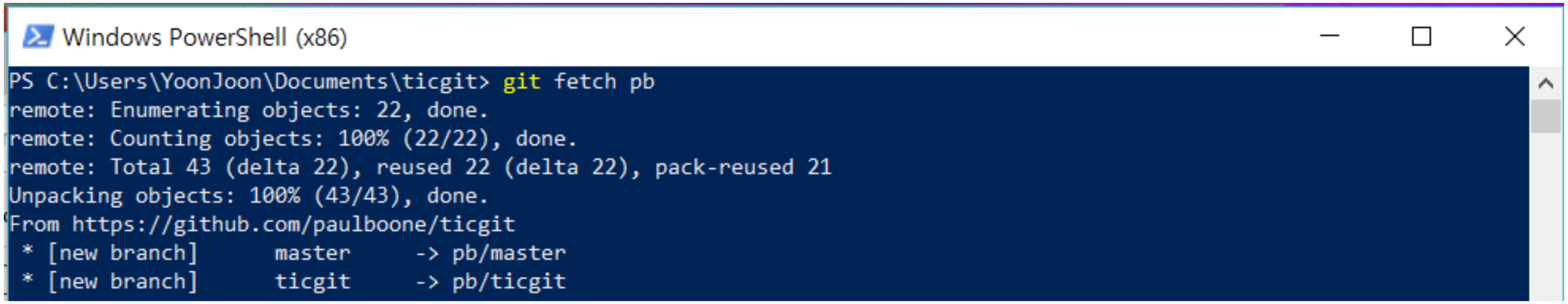
Adding Remotes

We've mentioned and given some demonstrations of how the git clone command implicitly adds the origin remote for us.



```
PS C:\Users\YoonJoon\Documents\ticgit> git remote
origin
PS C:\Users\YoonJoon\Documents\ticgit> git remote add pb https://github.com/paulboone/ticgit
>>
PS C:\Users\YoonJoon\Documents\ticgit> git remote -v
origin  https://github.com/schacon/ticgit (fetch)
origin  https://github.com/schacon/ticgit (push)
pb      https://github.com/paulboone/ticgit (fetch)
pb      https://github.com/paulboone/ticgit (push)
```

Now we can use the string **`pb`** on the command line in lieu of the whole URL.



```
PS C:\Users\YoonJoon\Documents\ticgit> git fetch pb
remote: Enumerating objects: 22, done.
remote: Counting objects: 100% (22/22), done.
remote: Total 43 (delta 22), reused 22 (delta 22), pack-reused 21
Unpacking objects: 100% (43/43), done.
From https://github.com/paulboone/ticgit
 * [new branch]      master     -> pb/master
 * [new branch]      ticgit     -> pb/ticgit
```

Paul's master branch is now accessible locally as **`pb/master`**.

Fetching and Pulling from Remotes

As we just saw, to get data from our remote projects, we can run:

```
$ git fetch <remote>
```

The command goes out to that remote project and pulls down all the data from that remote project that we don't have yet. After we do this, we should have references to all the branches from that remote, which we can merge in or inspect at any time.

- If we clone a repository, the command automatically adds that remote repository under the name “origin”. So, `git fetch origin` fetches any new work that has been pushed to that server since we cloned (or last fetched from) it.
- It’s important to note that the `git fetch` command only downloads the data to our local repository — it doesn’t automatically merge it with any of our work or modify what we’re currently working on. We have to merge it manually into our work when we’re ready.

- If our current branch is set up to track a remote branch, we can use the **`git pull`** command to automatically fetch and then merge that remote branch into our current branch.
- This may be an easier or more comfortable workflow; and by default, the **`git clone`** command automatically sets up your local master branch to track the remote master branch on the server we cloned from.
- Running **`git pull`** generally fetches data from the server we originally cloned from and automatically tries to merge it into the code we're currently working on.

Pushing to Remotes

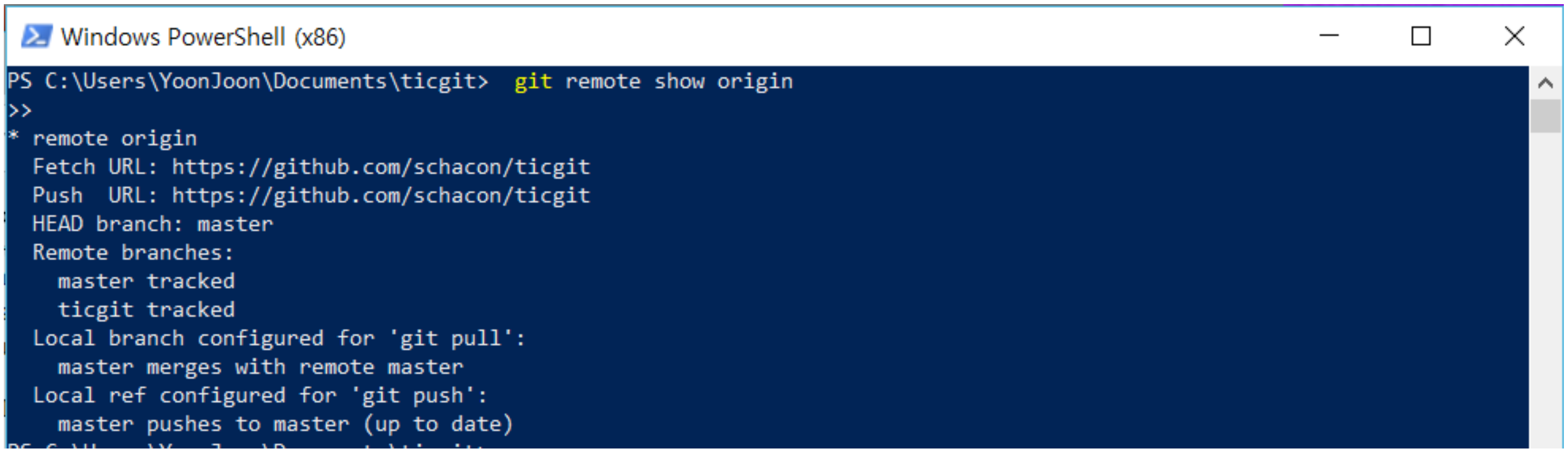
When we have our project at a point that we want to share, we have to push it upstream. The command for this is simple: `git push <remote> <branch>`.

```
$ git push origin master
```

- This command works only if we cloned from a server to which we have write access and if nobody has pushed in the meantime.
- If we and someone else clone at the same time and they push upstream and then we push upstream, our push will rightly be rejected.

Inspecting a Remote

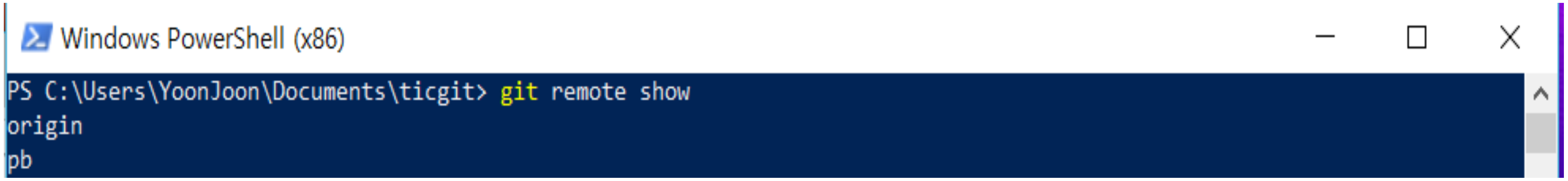
If we want to see more information about a particular remote, we can use the `git remote show <remote>` command.



Windows PowerShell (x86)

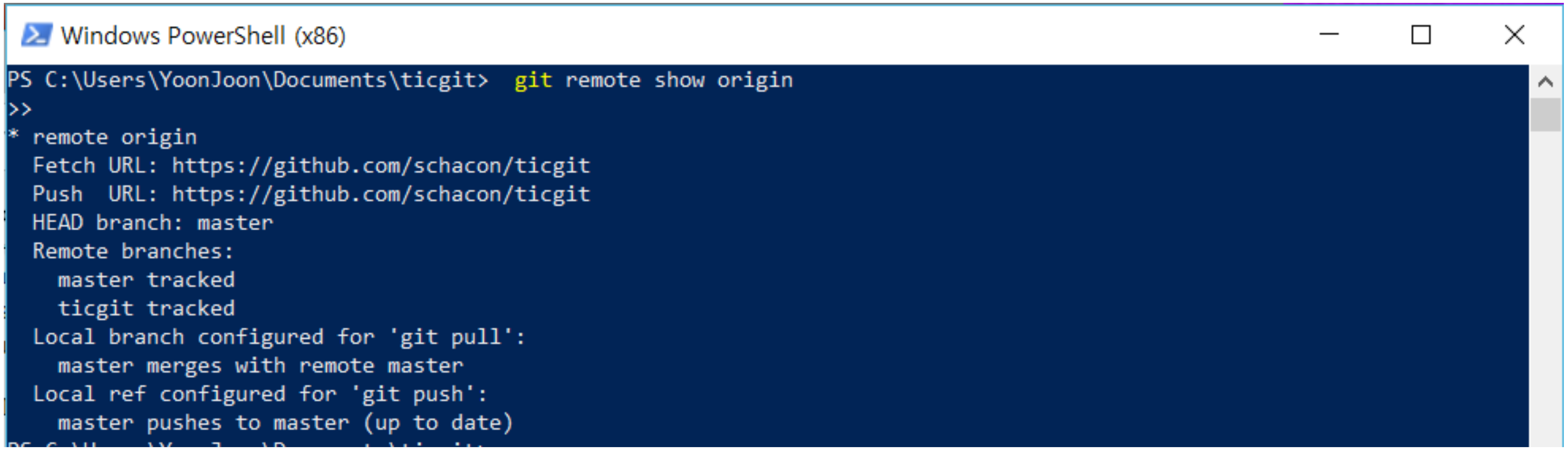
```
PS C:\Users\YoonJoon\Documents\ticgit>  git remote show origin
>>
* remote origin
  Fetch URL: https://github.com/schacon/ticgit
  Push  URL: https://github.com/schacon/ticgit
  HEAD branch: master
  Remote branches:
    master tracked
    ticgit tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

- It lists the URL for the remote repository as well as the tracking branch information.
- The command helpfully tells us that if we're on the master branch and we run `git pull`, it will automatically merge in the master branch on the remote after it fetches all the remote references.
- It also lists all the remote references it has pulled down.



Windows PowerShell (x86)

```
PS C:\Users\YoonJoon\Documents\ticgit> git remote show
origin
pb
```



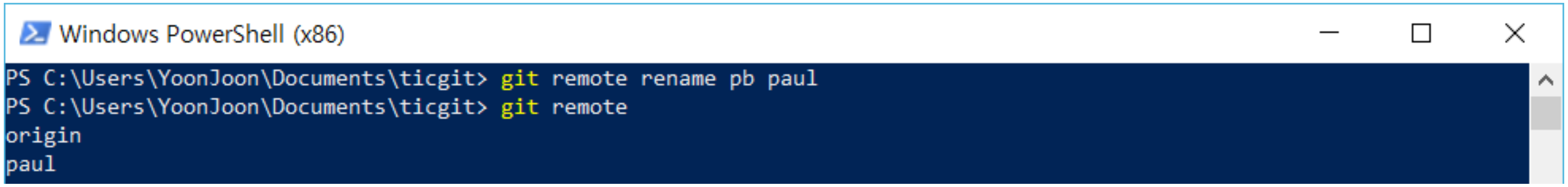
Windows PowerShell (x86)

```
PS C:\Users\YoonJoon\Documents\ticgit>  git remote show origin
>>
* remote origin
  Fetch URL: https://github.com/schacon/ticgit
  Push  URL: https://github.com/schacon/ticgit
  HEAD branch: master
  Remote branches:
    master tracked
    ticgit tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

- This command shows which branch is automatically pushed to when we run `git push` while on certain branches.
- It also shows us which remote branches on the server you don't yet have, which remote branches we have that have been removed from the server, and multiple local branches that are able to merge automatically with their remote-tracking branch when we run `git pull`.

Renaming and Removing Remote

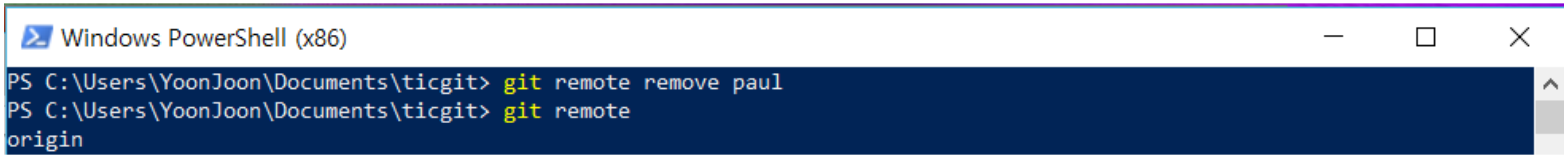
We can run `git remote rename` to change a remote's shortname.



```
PS C:\Users\YoonJoon\Documents\ticgit> git remote rename pb paul
PS C:\Users\YoonJoon\Documents\ticgit> git remote
origin
paul
```

It's worth mentioning that this changes all our remote-tracking branch names, too. What used to be referenced at `pb/master` is now at `paul/master`.

If you want to remove a remote for some reason we can either use `git remote remove` or `git remote rm`:



Windows PowerShell (x86)

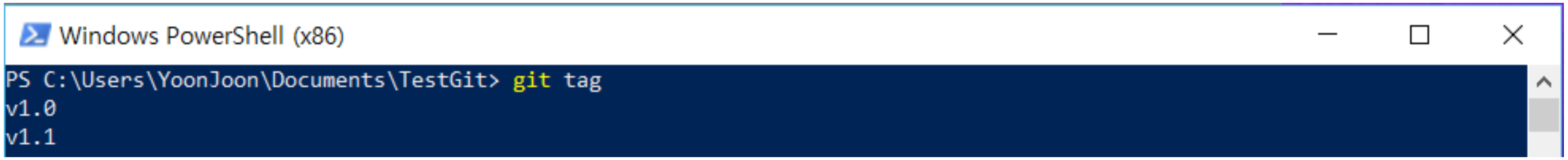
```
PS C:\Users\YoonJoon\Documents\ticgit> git remote remove paul
PS C:\Users\YoonJoon\Documents\ticgit> git remote
origin
```

Tagging

Git has the ability to tag specific points in history as being important. Typically people use this functionality to mark release points (v1.0, and so on).

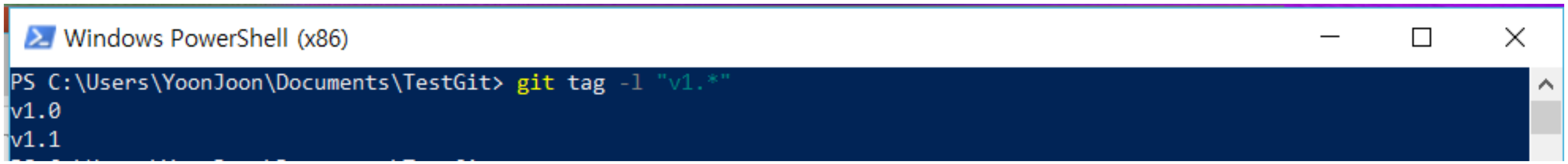
Listing Your Tags

Listing the available tags in Git is straightforward. Just type `git tag` (with optional `-l` or `--list`):



Windows PowerShell (x86)

```
PS C:\Users\YoonJoon\Documents\TestGit> git tag
v1.0
v1.1
```



Windows PowerShell (x86)

```
PS C:\Users\YoonJoon\Documents\TestGit> git tag -l "v1.*"
v1.0
v1.1
```

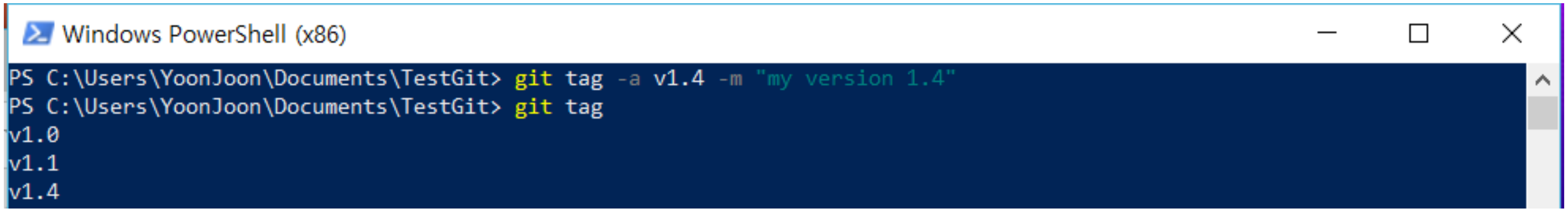
Creating Tags

Git supports two types of tags: *lightweight* and *annotated*.

- A lightweight tag is very much like a branch that doesn’t change — it’s just a pointer to a specific commit.
- Annotated tags, however, are stored as full objects in the Git database. They’re checksummed; contain the tagger name, email, and date; have a tagging message; and can be signed and verified with GNU Privacy Guard (GPG).

Annotated Tags

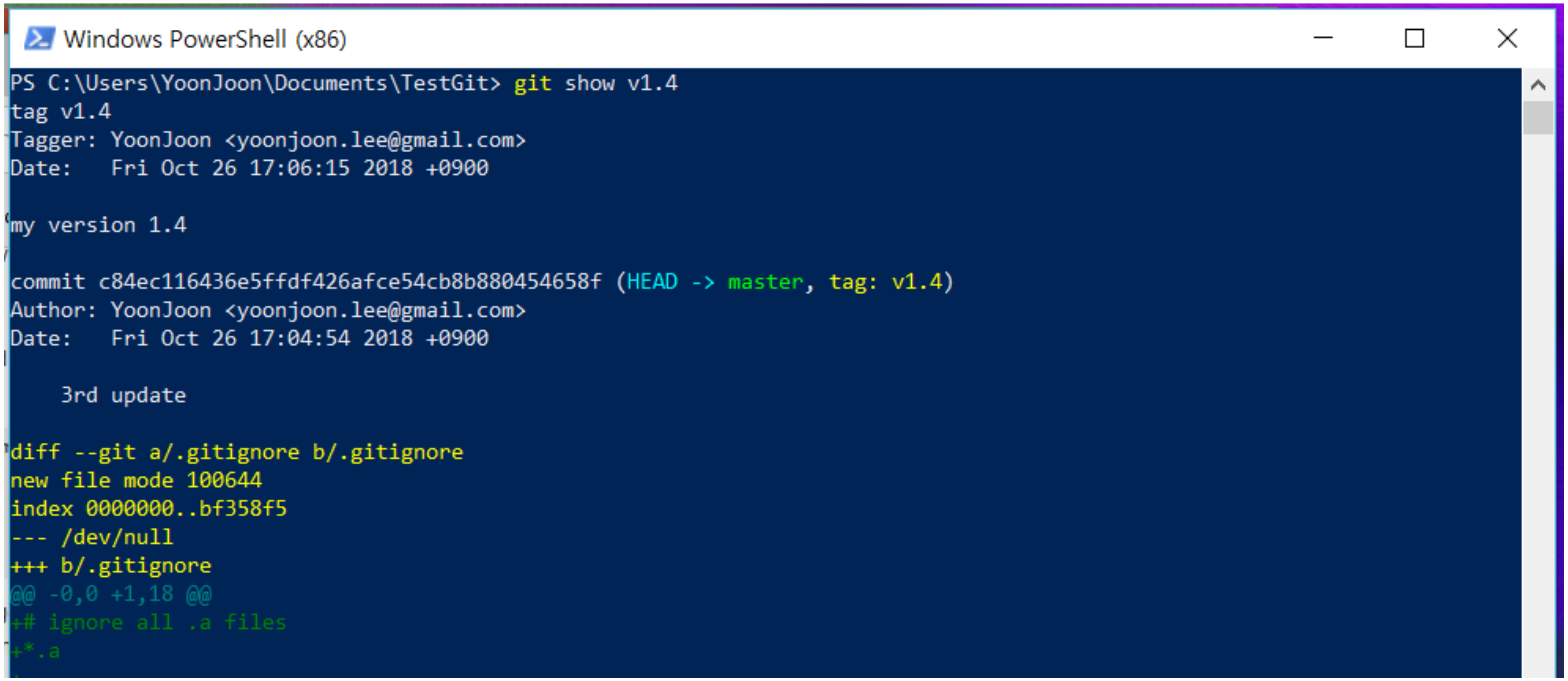
Creating an annotated tag in Git is simple. The easiest way is to specify `-a` when you run the `tag` command:



```
PS C:\Users\YoonJoon\Documents\TestGit> git tag -a v1.4 -m "my version 1.4"
PS C:\Users\YoonJoon\Documents\TestGit> git tag
v1.0
v1.1
v1.4
```

The `-m` specifies a tagging message, which is stored with the tag.

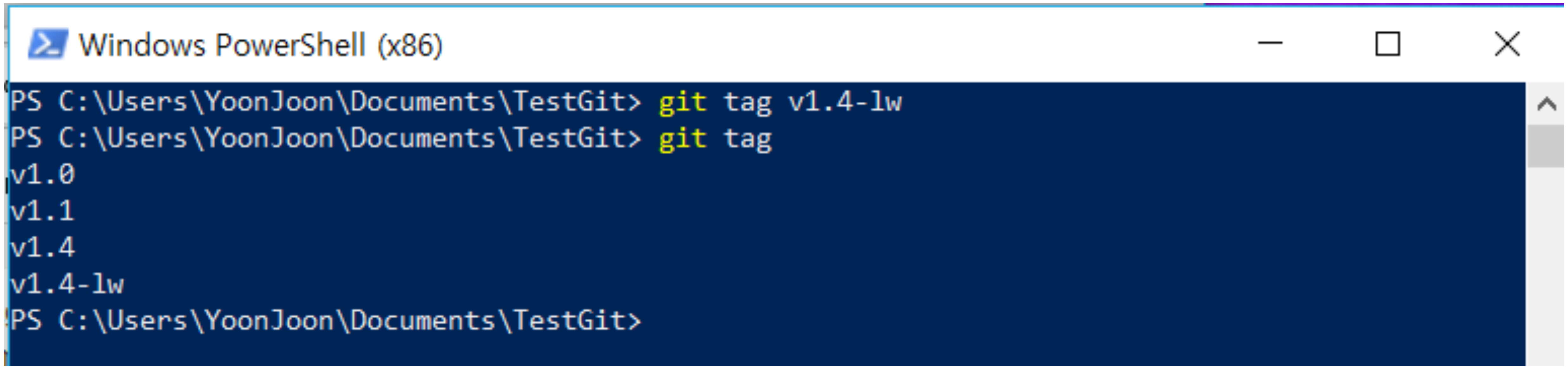
You can see the tag data along with the commit that was tagged by using the `git show` command:



```
PS C:\Users\YoonJoon\Documents\TestGit> git show v1.4
tag v1.4
Tagger: YoonJoon <yoonjoon.lee@gmail.com>
Date:   Fri Oct 26 17:06:15 2018 +0900

my version 1.4

commit c84ec116436e5ffdf426afce54cb8b880454658f (HEAD -> master, tag: v1.4)
Author: YoonJoon <yoonjoon.lee@gmail.com>
Date:   Fri Oct 26 17:04:54 2018 +0900

    3rd update

diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..bf358f5
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1,18 @@
+# ignore all .a files
+*.a
```

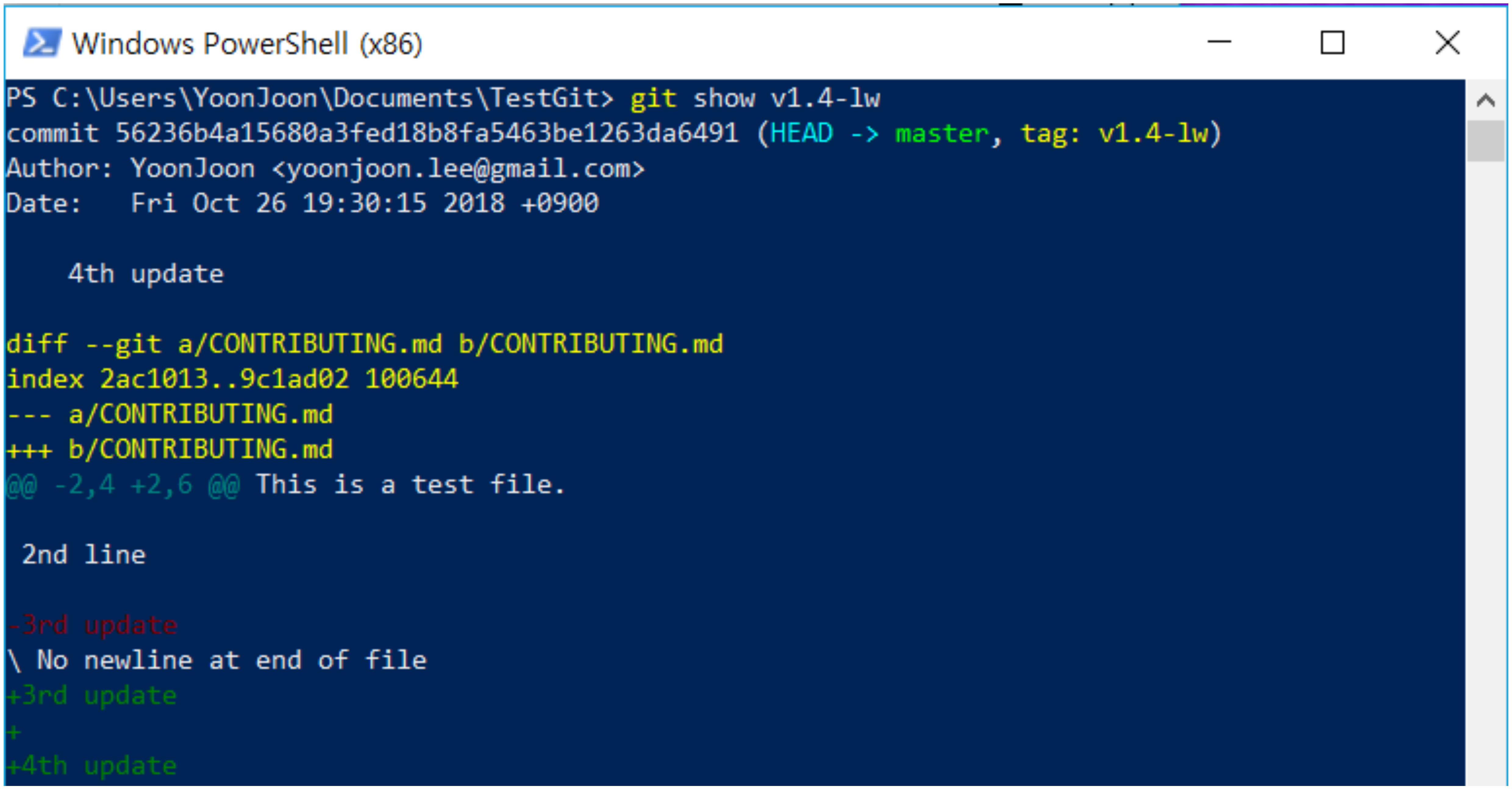
Lightweight Tags

- Another way to tag commits is with a lightweight tag.
- This is basically the commit checksum stored in a file — no other information is kept.
- To create a lightweight tag, don't supply any of the `-a`, `-s`, or `-m` options, just provide a tag name:



```
PS C:\Users\YoonJoon\Documents\TestGit> git tag v1.4-lw
PS C:\Users\YoonJoon\Documents\TestGit> git tag
v1.0
v1.1
v1.4
v1.4-lw
PS C:\Users\YoonJoon\Documents\TestGit>
```

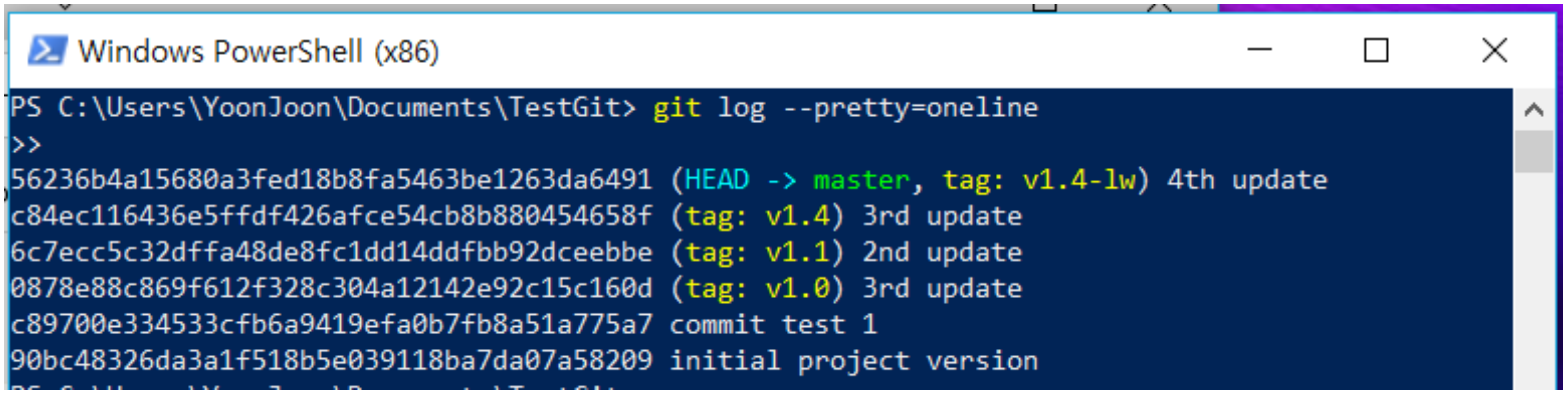
If we run `git show` on the tag, we don't see the extra tag information. The command just shows the commit:



```
PS C:\Users\YoonJoon\Documents\TestGit> git show v1.4-lw
commit 56236b4a15680a3fed18b8fa5463be1263da6491 (HEAD -> master, tag: v1.4-lw)
Author: YoonJoon <yoonjoon.lee@gmail.com>
Date:   Fri Oct 26 19:30:15 2018 +0900

    4th update

diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
index 2ac1013..9c1ad02 100644
--- a/CONTRIBUTING.md
+++ b/CONTRIBUTING.md
@@ -2,4 +2,6 @@ This is a test file.

 2nd line

-3rd update
\ No newline at end of file
+3rd update
+
+4th update
```

Tagging Later

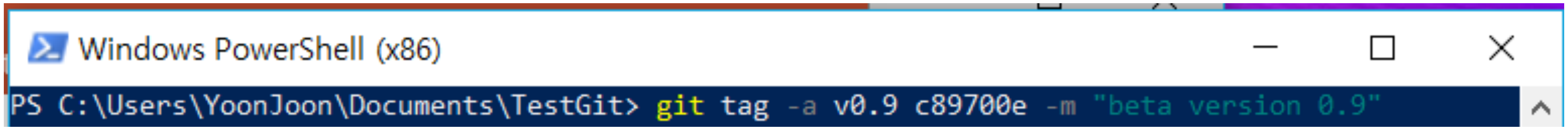
We can also tag commits after we've moved past them. Suppose your commit history looks like this:



Windows PowerShell (x86)

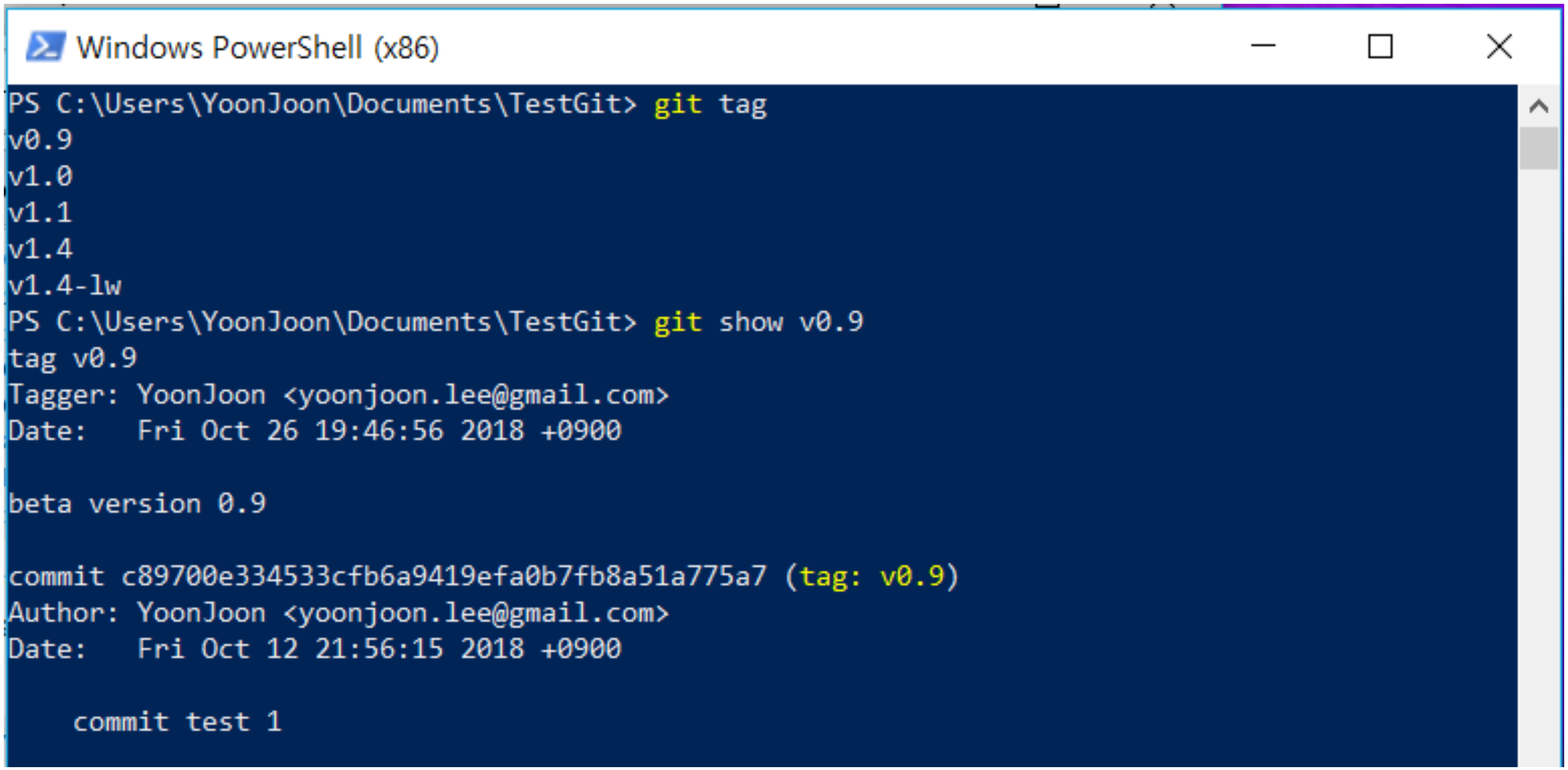
```
PS C:\Users\YoonJoon\Documents\TestGit> git log --pretty=oneline
>>
56236b4a15680a3fed18b8fa5463be1263da6491 (HEAD -> master, tag: v1.4-lw) 4th update
c84ec116436e5ffdf426afce54cb8b8804546S8f (tag: v1.4) 3rd update
6c7ecc5c32dffa48de8fc1dd14ddfbb92dceebbe (tag: v1.1) 2nd update
0878e88c869f612f328c304a12142e92c15c160d (tag: v1.0) 3rd update
c89700e334533cfb6a9419efa0b7fb8a51a775a7 commit test 1
90bc48326da3a1f518b5e039118ba7da07a58209 initial project version
```

Suppose we forgot to tag the project at v0.9, which was at the "`commit test 1`" commit. We can add it after the fact.



Windows PowerShell (x86)

```
PS C:\Users\YoonJoon\Documents\TestGit> git tag -a v0.9 c89700e -m "beta version 0.9"
```

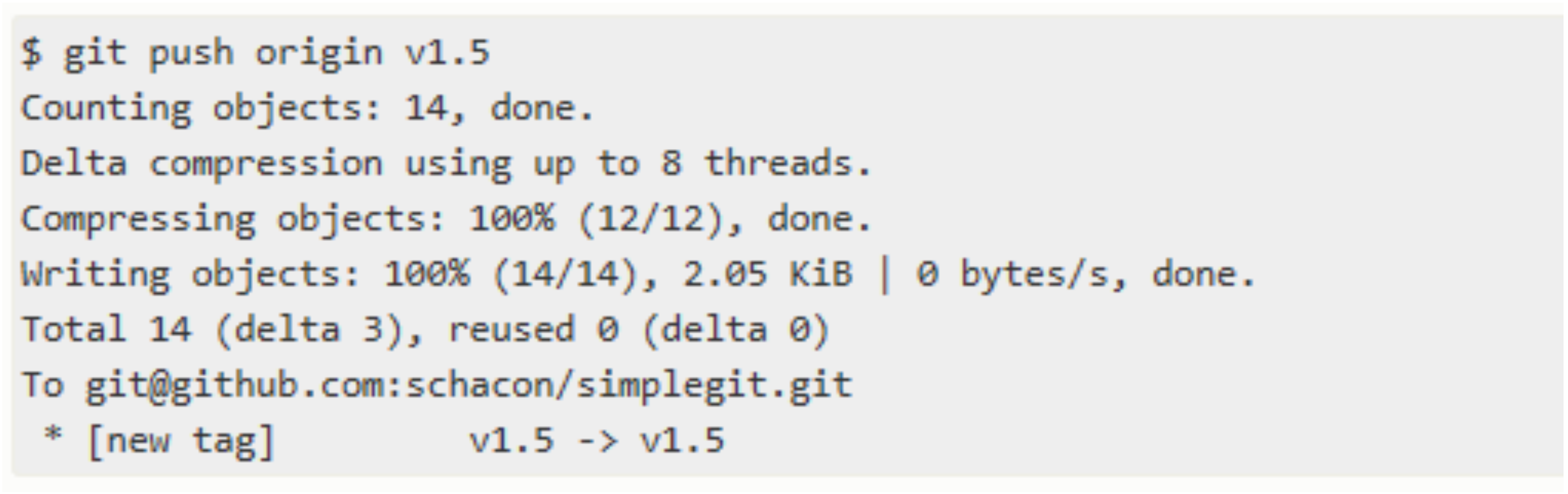


Windows PowerShell (x86)

```
PS C:\Users\YoonJoon\Documents\TestGit> git tag
v0.9
v1.0
v1.1
v1.4
v1.4-lw
PS C:\Users\YoonJoon\Documents\TestGit> git show v0.9
tag v0.9
Tagger: YoonJoon <yoonjoon.lee@gmail.com>
Date:   Fri Oct 26 19:46:56 2018 +0900

beta version 0.9

commit c89700e334533cfb6a9419efa0b7fb8a51a775a7 (tag: v0.9)
Author: YoonJoon <yoonjoon.lee@gmail.com>
Date:   Fri Oct 12 21:56:15 2018 +0900

    commit test 1
```

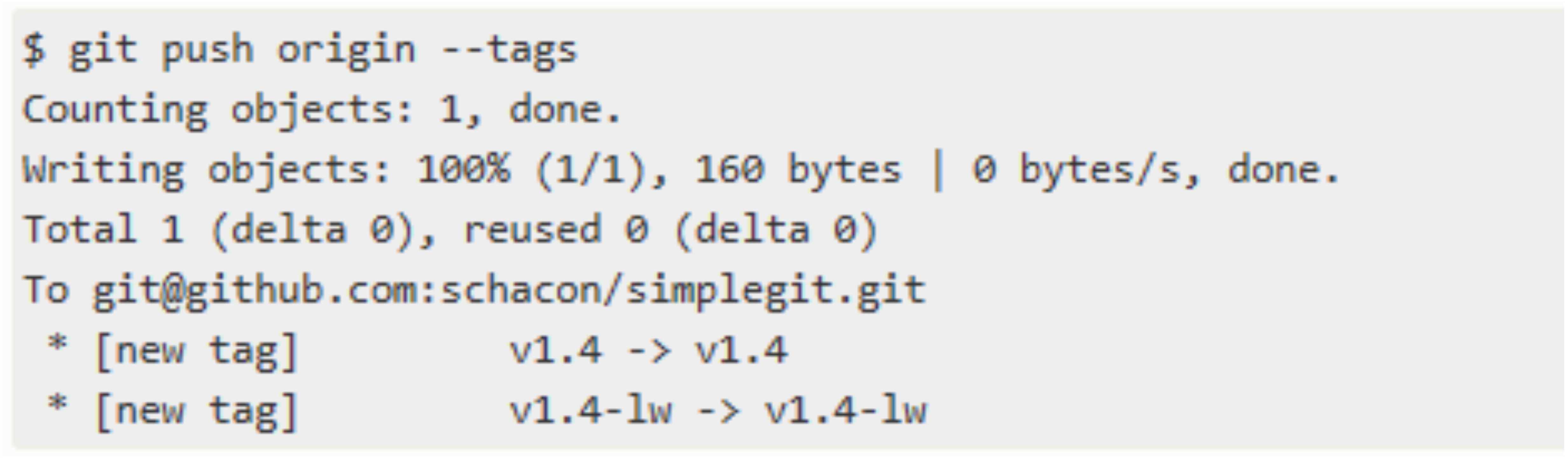
Sharing Tags

By default, the `git push` command doesn't transfer tags to remote servers. We will have to explicitly push tags to a shared server after you have created them. This process is just like sharing remote branches — we can run `git push origin <tagname>`.



```
$ git push origin v1.5
Counting objects: 14, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (12/12), done.
Writing objects: 100% (14/14), 2.05 KiB | 0 bytes/s, done.
Total 14 (delta 3), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new tag]         v1.5 -> v1.5
```

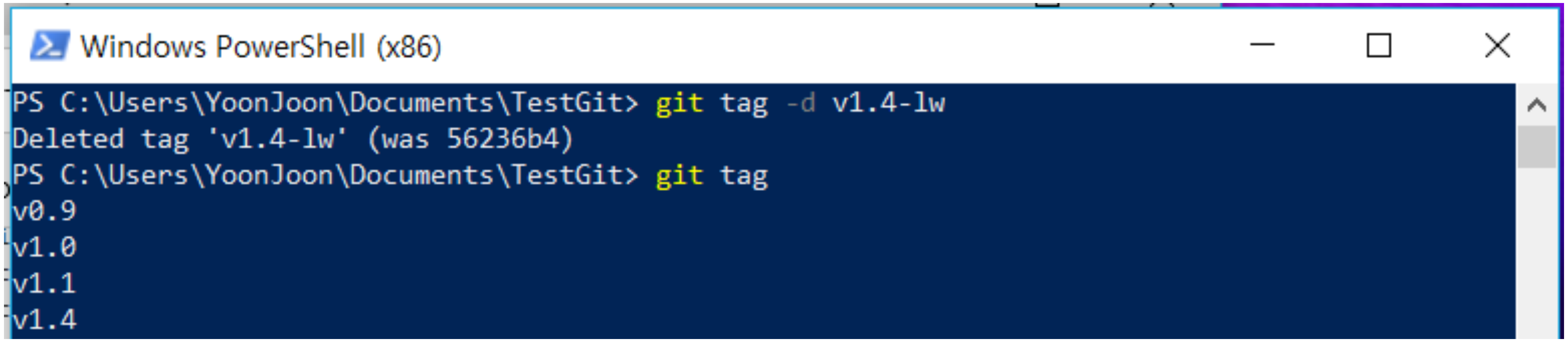
If we have a lot of tags that we want to push up at once, we can also use the `--tags` option to the `git push` command. This will transfer all of our tags to the remote server that are not already there.



```
$ git push origin --tags
Counting objects: 1, done.
Writing objects: 100% (1/1), 160 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new tag]         v1.4 -> v1.4
 * [new tag]         v1.4-lw -> v1.4-lw
```

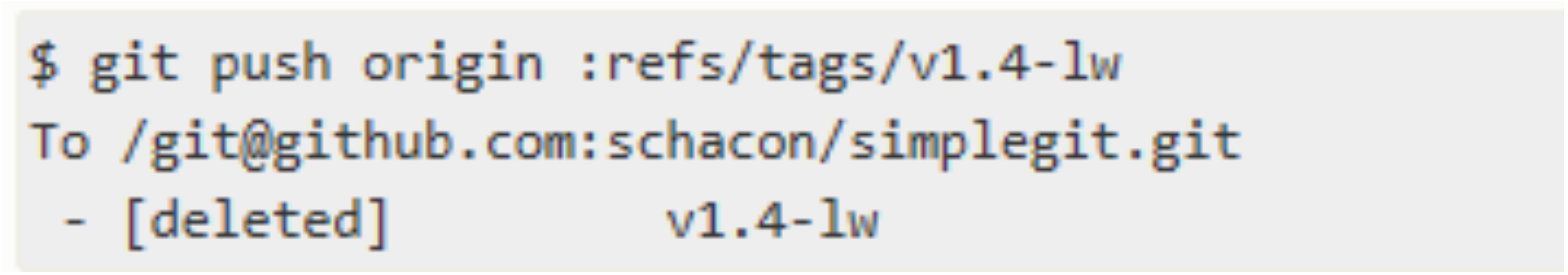
Deleting Tags

To delete a tag on our local repository, we can use `git tag -d <tagname>`. For example, we could remove our lightweight tag above as follows:



```
PS C:\Users\YoonJoon\Documents\TestGit> git tag -d v1.4-lw
Deleted tag 'v1.4-lw' (was 56236b4)
PS C:\Users\YoonJoon\Documents\TestGit> git tag
v0.9
v1.0
v1.1
v1.4
```

Note that this does not remove the tag from any remote servers. In order to update any remotes, we must use `git push <remote> :refs/tags/<tagname>`:



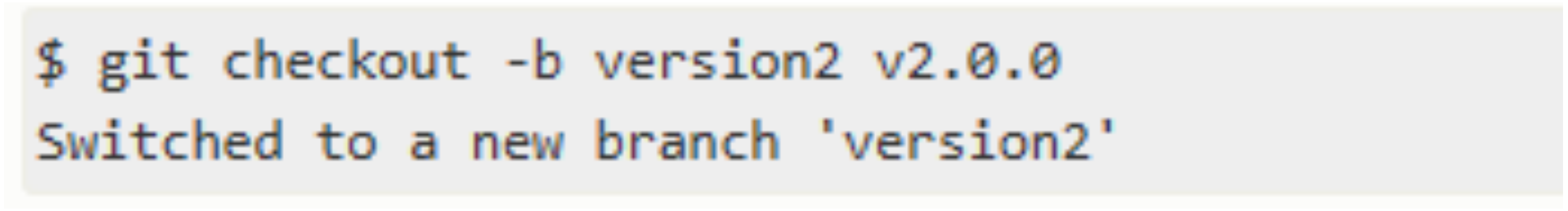
```
$ git push origin :refs/tags/v1.4-lw
To /git@github.com:schacon/simplegit.git
 - [deleted]         v1.4-lw
```

Checking out Tags

If we want to view the versions of files a tag is pointing to, we can do a `git checkout`, though this puts our repository in “detached HEAD” state, which has some ill side effects:

```
$ git checkout 2.0.0
Note: checking out '2.0.0'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch>

HEAD is now at 99ada87... Merge pull request #89 from schacon/appendix-final

$ git checkout 2.0-beta-0.1
Previous HEAD position was 99ada87... Merge pull request #89 from schacon/appendix-final
HEAD is now at df3f601... add atlas.json and cover image
```

In “detached HEAD” state, if we make changes and then create a commit, the tag will stay the same, but our new commit won’t belong to any branch and will be unreachable, except by the exact commit hash. Thus, if we need to make changes — say we’re fixing a bug on an older version, for instance — we will generally want to create a branch:

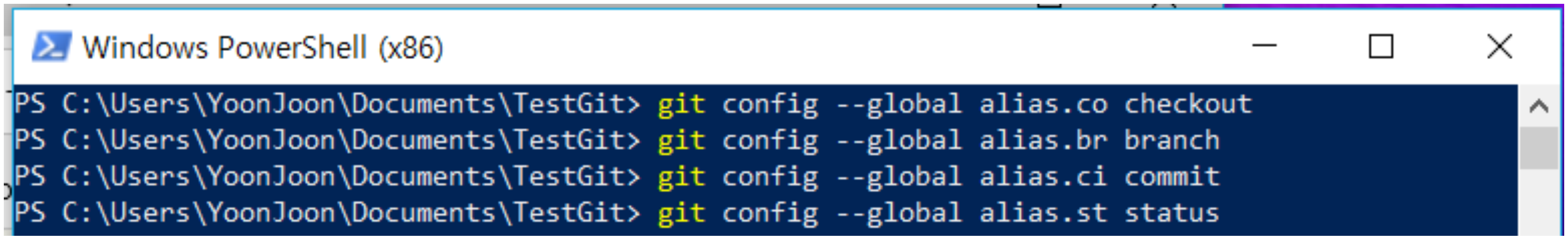


```
$ git checkout -b version2 v2.0.0
Switched to a new branch 'version2'
```

If we do this and make a commit,
our version2 branch will be slightly different than our v2.0.0 tag since it will move forward with our new changes, so do be careful.

Git Aliases

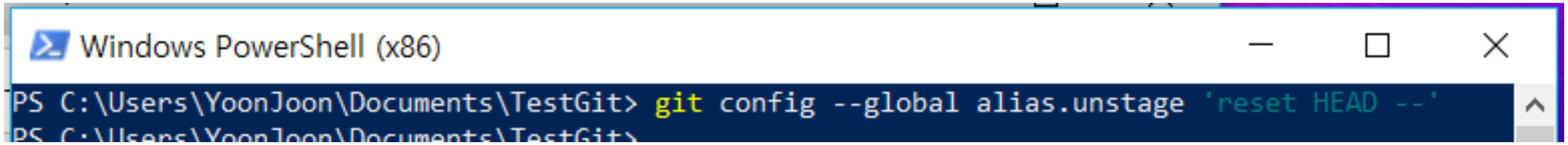
- There's just one little tip that can make our Git experience simpler, easier, and more familiar: aliases.
- If we don't want to type the entire text of each of the Git commands, we can easily set up an alias for each command using `git config`.



```
PS C:\Users\YoonJoon\Documents\TestGit> git config --global alias.co checkout
PS C:\Users\YoonJoon\Documents\TestGit> git config --global alias.br branch
PS C:\Users\YoonJoon\Documents\TestGit> git config --global alias.ci commit
PS C:\Users\YoonJoon\Documents\TestGit> git config --global alias.st status
```

This means that, for example, instead of typing `git commit`, we just need to type `git ci`.

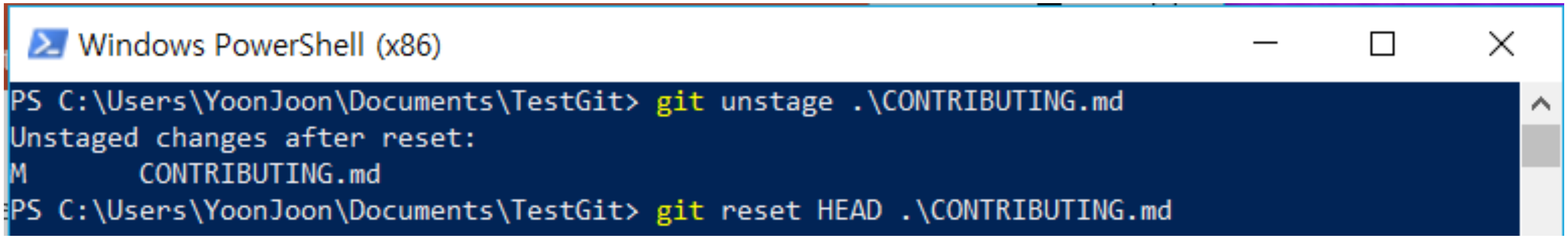
This technique can also be very useful in creating commands that we think should exist.



Windows PowerShell (x86)

```
PS C:\Users\YoonJoon\Documents\TestGit> git config --global alias.unstage 'reset HEAD --'
PS C:\Users\YoonJoon\Documents\TestGit>
```

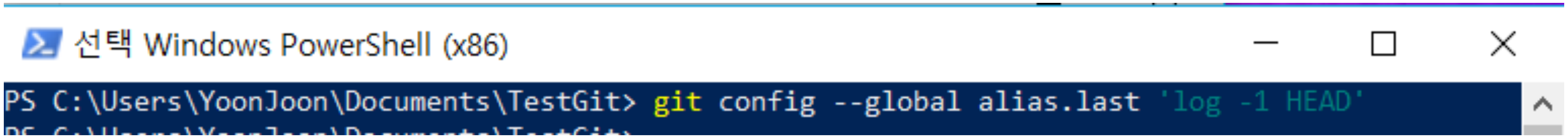
This makes the following two commands equivalent:



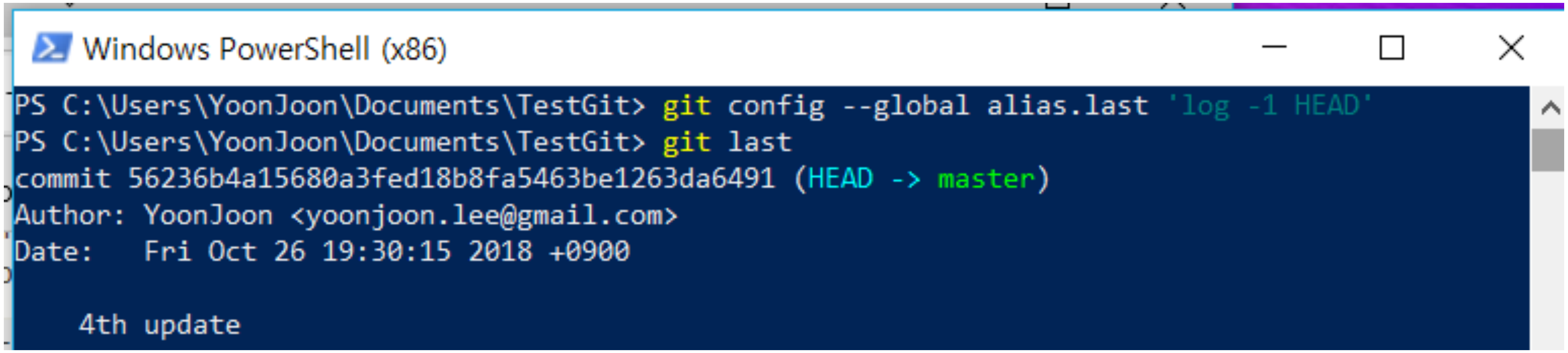
Windows PowerShell (x86)

```
PS C:\Users\YoonJoon\Documents\TestGit> git unstage .\CONTRIBUTING.md
Unstaged changes after reset:
M       CONTRIBUTING.md
PS C:\Users\YoonJoon\Documents\TestGit> git reset HEAD .\CONTRIBUTING.md
```

This seems a bit clearer. It's also common to add a `last` command, like this:

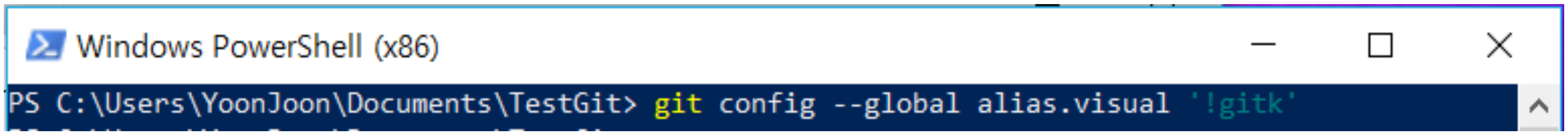

선택 Windows PowerShell (x86)

```
PS C:\Users\YoonJoon\Documents\TestGit> git config --global alias.last 'log -1 HEAD'
```

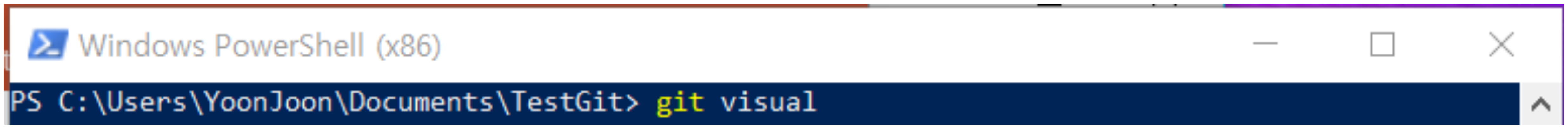

Windows PowerShell (x86)

```
PS C:\Users\YoonJoon\Documents\TestGit> git config --global alias.last 'log -1 HEAD'
PS C:\Users\YoonJoon\Documents\TestGit> git last
commit 56236b4a15680a3fed18b8fa5463be1263da6491 (HEAD -> master)
Author: YoonJoon <yoonjoon.lee@gmail.com>
Date:   Fri Oct 26 19:30:15 2018 +0900

    4th update
```

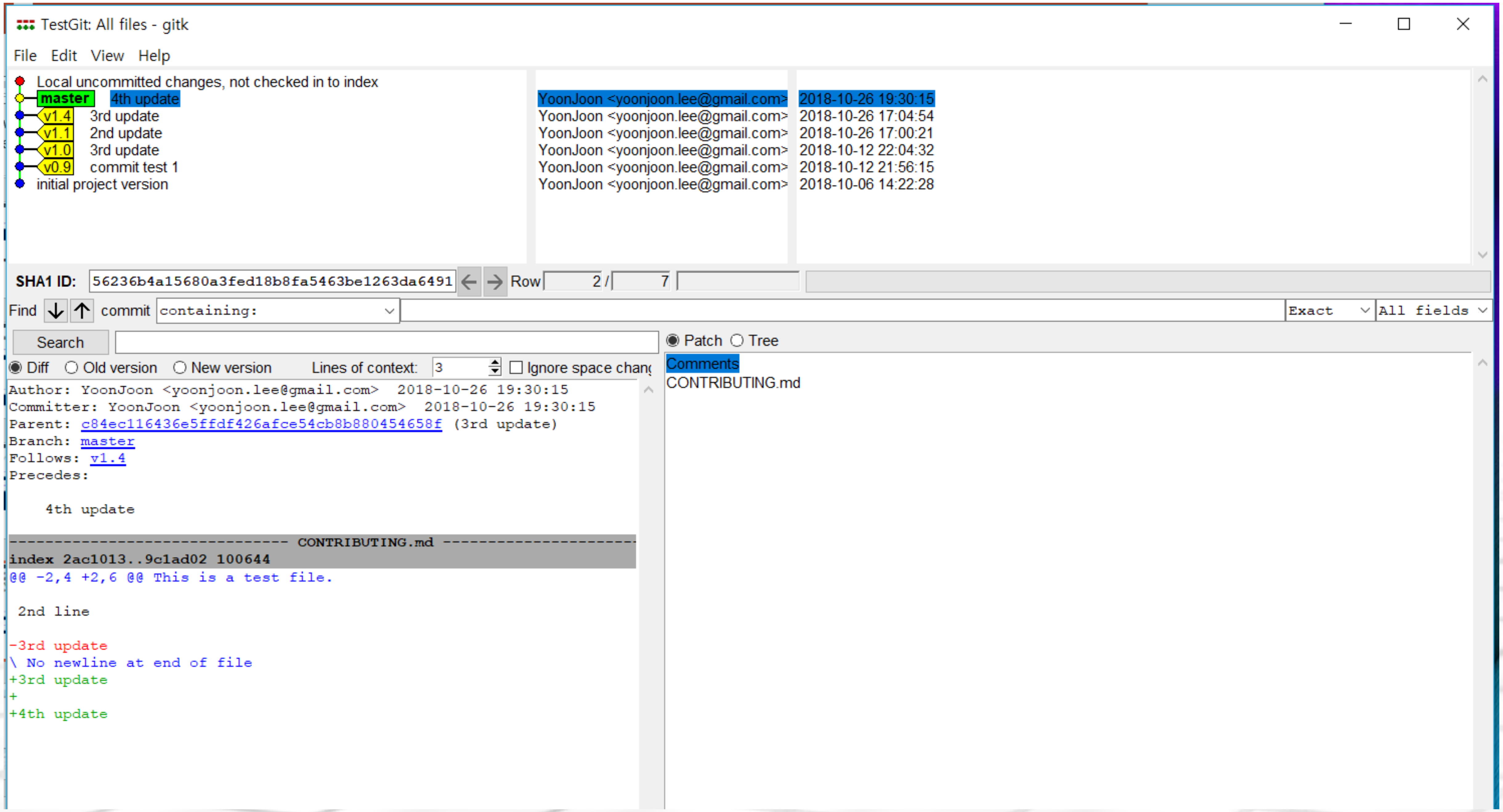
Maybe we want to run an external command, rather than a Git subcommand. In that case, we start the command with a `!` character. This is useful if we write our own tools that work with a Git repository. We can demonstrate by aliasing `git visual` to run `gitk`:



```
PS C:\Users\YoonJoon\Documents\TestGit> git config --global alias.visual '!gitk'
```



```
PS C:\Users\YoonJoon\Documents\TestGit> git visual
```



TestGit: All files - gitk

File Edit View Help

```
Local uncommitted changes, not checked in to index
master  4th update           YoonJoon <yoonjoon.lee@gmail.com>  2018-10-26 19:30:15
v1.4  3rd update             YoonJoon <yoonjoon.lee@gmail.com>  2018-10-26 17:04:54
v1.1  2nd update             YoonJoon <yoonjoon.lee@gmail.com>  2018-10-26 17:00:21
v1.0  3rd update             YoonJoon <yoonjoon.lee@gmail.com>  2018-10-12 22:04:32
v0.9  commit test 1          YoonJoon <yoonjoon.lee@gmail.com>  2018-10-12 21:56:15
initial project version      YoonJoon <yoonjoon.lee@gmail.com>  2018-10-06 14:22:28
```

SHA1 ID: 56236b4a15680a3fed18b8fa5463be1263da6491 Row 2 / 7

Find commit containing: Exact All fields

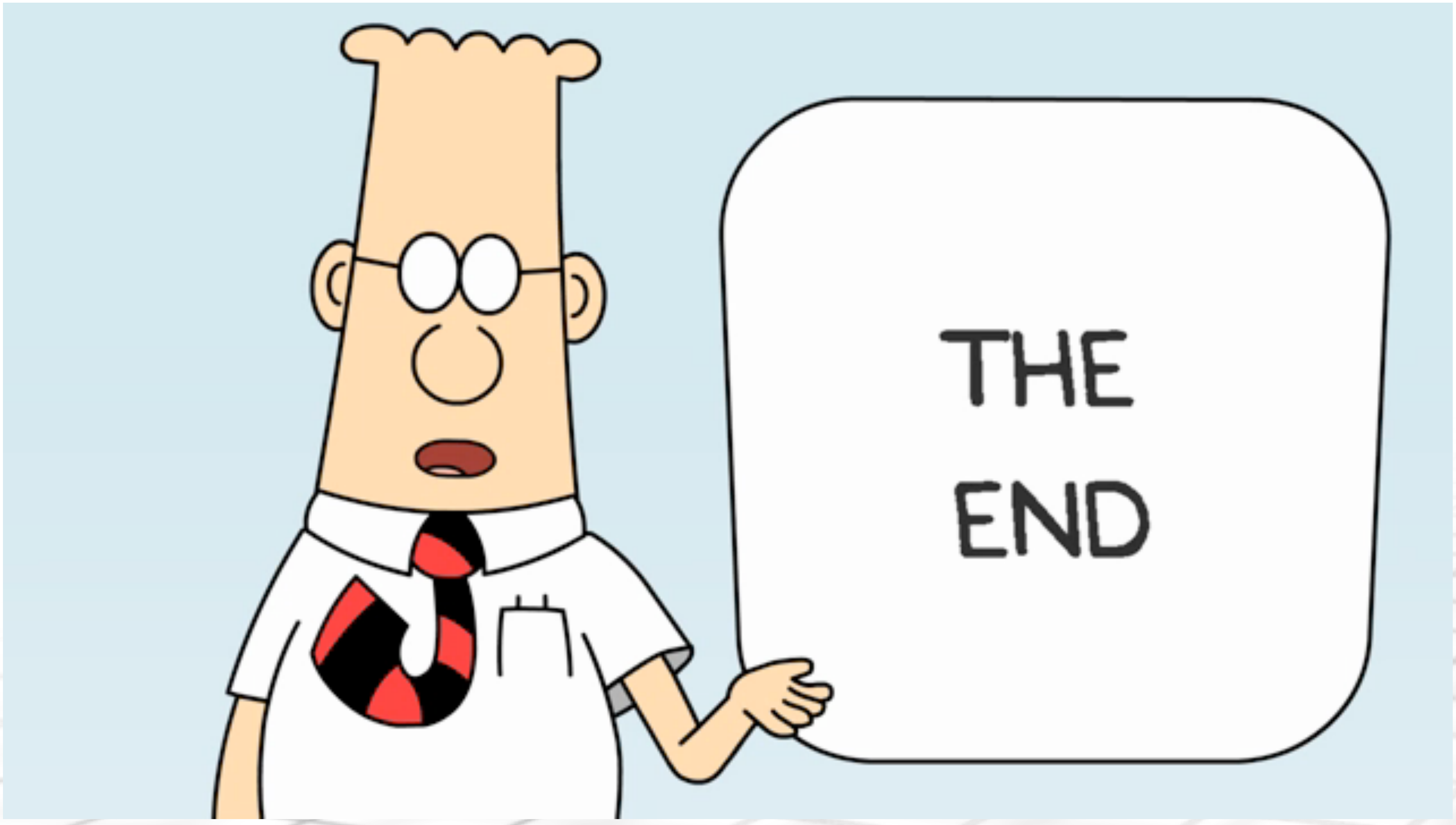
Search

Diff ○ Old version ○ New version Lines of context: 3 ☐ Ignore space chang

● Patch ○ Tree

Comments
CONTRIBUTING.md

```
Author: YoonJoon <yoonjoon.lee@gmail.com>  2018-10-26 19:30:15
Committer: YoonJoon <yoonjoon.lee@gmail.com>  2018-10-26 19:30:15
Parent: c84ec116436e5ffdf426afce54cb8b880454658f (3rd update)
Branch: master
Follows: v1.4
Precedes:

    4th update

------------------------------- CONTRIBUTING.md ----------------------
index 2ac1013..9c1ad02 100644
@@ -2,4 +2,6 @@ This is a test file.

 2nd line

-3rd update
\ No newline at end of file
+3rd update
+
+4th update
```



출처: metachannels.com

감사합니다