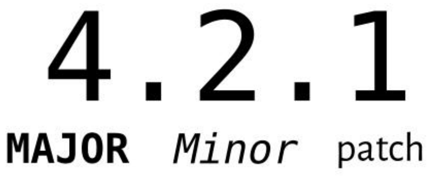
Git/GitHub versions, tags, and releases
Software Engineering for Scientists

Once you have reached a particular milestone with your software---the software has the features you intended and its operation is stable---you may want to indicate that milestone with a version number and make that version publicly available (a la Python 3.8.6 or Numpy 1.19.4). However, if you are continuing to develop your software then you will want this version to integrate seamlessly into your commit history. For this purpose git provides the `tag` function:

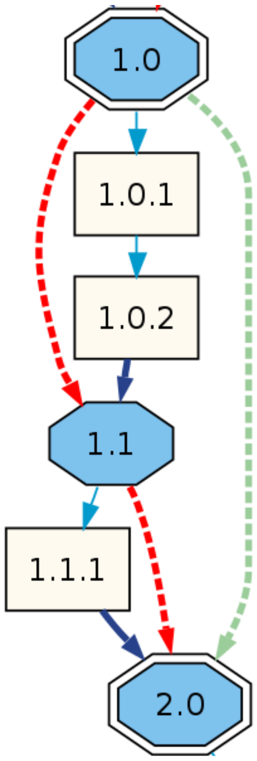
```
usage: git tag [-a | -s | -u <key-id>] [-f] [-m <msg> | -F <file>]
                <tagname> [<head>]
   or: git tag -d <tagname>...
   or: git tag -l [-n[<num>]] [--contains <commit>] [--no-contains <commit>]
[--points-at <object>]
                [--format=<format>] [--[no-]merged [<commit>]] [<pattern>...]
   or: git tag -v [--format=<format>] <tagname>...

    -l, --list            list tag names
    -n[<n>]               print <n> lines of each tag message
    -d, --delete          delete tags
    -v, --verify          verify tags

Tag creation options
    -a, --annotate        annotated tag, needs a message
    -m, --message <message>
                          tag message
    -F, --file <file>     read message from file
    -e, --edit            force edit of tag message
    -s, --sign            annotated and GPG-signed tag
    --cleanup <mode>      how to strip spaces and #comments from message
    -u, --local-user <key-id>
                          use another key to sign the tag
    -f, --force           replace the tag if exists
    --create-reflog       create a reflog

Tag listing options
    --column[=<style>]    show tag list in columns
    --contains <commit>   print only tags that contain the commit
    --no-contains <commit>
                          print only tags that don't contain the commit
    --merged <commit>     print only tags that are merged
    --no-merged <commit>  print only tags that are not merged
    --sort <key>          field name to sort on
    --points-at <object>  print only tags of the object
    --format <format>     format to use for the output
    --color[=<when>]      respect format colors
    -i, --ignore-case     sorting and filtering are case insensitive
```

Version naming convention

Git does not limit the format of your tags. You could tag your various versions with the names of trees or the names of your various goldfish. However, for the sake of consistency, most software uses a sequential, numeric, three-part versioning paradigm referred to as "semantic versioning" [wiki]. The first number represents a "major" release, the second number represents a "minor" release, and the third number represents a "patch."



What constitutes a major or minor release or a patch is subjective, but ultimately it's up to the dev team to decide, based on their judgement of the future trajectory of the software's development.



(Jacob's) rules of thumb on version naming:

1. **Major release:** Contains large changes to the features or functionality of the software that break functionality existing in previous versions.
2. **Minor release:** Contains added features/functionality that do not break functionality from previous versions of the software
3. **Patches:** Does not add new functionality but will fix bugs discovered in the existing features.

Consideration of where to delineate major, minor, and patch versions will depend on a lot of factors including severity of bugs, demand for new features, and timeliness of your changes. For example, many minor releases could have instead been rolled into a major release, later down the line, but this means those new features will take longer to be available to end-users. Ultimate, it is up to the developers' judgement. In this class we will be tagging our versions using the git/GitHub API.

Version tagging with Git

There are essentially two types of Git tags you can add, one is a "lightweight" tag, which does not contain any annotation info about the version, and the other is an "annotated" tag (uses the `-a` flag) which will store with the tag additional information like date, a message where you can log info about the version, and the email/name of the tagger.

It is recommended that you always use the “annotated” tag for any version of your software.

To add a tag using git, navigate to the repo and branch you want to tag and run the following:

```
git tag -a <version> -m “My version message.”
```

Here the <version> is the semantic version number. The version message should be something informative about what this version contains---what features were added, what previous features are no longer functional, etc. Feel free to make the message long and detailed. If you want to write a long message, don’t specify the message and git will launch a text editor for you to write it in.

As an example, in the below we have tagged the branch “`branch3`” in repo `swefs_test1` with the version 1.0.0. And we can see by running `git tag` that this is the only version for this repo:

```
(base) jovyan@jupyter-jast1849:~/swefs/sefs_test1$ git branch
* branch3
main
(base) jovyan@jupyter-jast1849:~/swefs/sefs_test1$ git tag -a 1.0.0 -m
"Releasing version 1.0.0"
(base) jovyan@jupyter-jast1849:~/swefs/sefs_test1$ git tag
1.0.0
```

To look at the details of the annotation, you use git show. You will see information about the tagger, the date, the tag message, and then the information about the associated commit:

```
(base) jovyan@jupyter-jast1849:~/swefs/sefs_test1$ git show 1.0.0
tag 1.0.0
Tagger: jtstanley <jtstan@gmail.com>
Date:   Fri Aug 13 19:42:10 2021 +0000

Releasing version 1.0.0

commit 88d287db237bf7808e237fe8187895fc182eedbe (HEAD -> branch3, tag: 1.0.0,
origin/branch3)
Author: jtstanley <jtstan@gmail.com>
Date:   Wed Aug 11 22:26:54 2021 +0000

    Added another line to file3 when I shouldn't have?

diff --git a/file3.txt b/file3.txt
index f37806a..b18307e 100644
--- a/file3.txt
+++ b/file3.txt
@@ -1 +1,3 @@
 Third file!
+
+I'll add a second line after my merge...
(base) jovyan@jupyter-jast1849:~/swefs/sefs_test1$
```

To view the state of the files corresponding to a version, use `git checkout`:

```
(base) jovyan@jupyter-jast1849:~/swefs/sefs_test1$ git checkout 1.0.0
Note: switching to '1.0.0'.

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 switching back to a branch.

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

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 88d287d Added another line to file3 when I shouldn't have?
```

NOTE: make sure that you do not continue development in the “detached HEAD” state. The commits you make are not associated with any branch and will be unreachable. To continue development you need to git checkout the desired branch, moving you off of the tag version.

Sharing tags

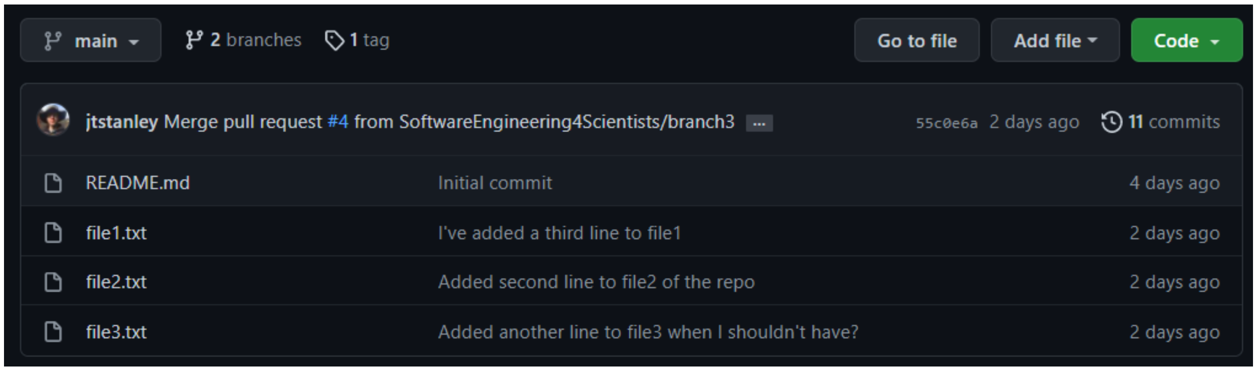
In order to make this version tag available, you will need to push the tag to the remote repo. This is done essentially in the same manner as pushing a local branch. To do so, run:

```
git push origin <tagname>
```

Here we see the new tag 1.0.0 has been created on the remote repository:

```
(base) jovyan@jupyter-jast1849:~/swefs/sefs_test1$ git push origin 1.0.0
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 162 bytes | 81.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To github.com:SoftwareEngineering4Scientists/sefs_test1.git
 * [new tag]         1.0.0 -> 1.0.0
(base) jovyan@jupyter-jast1849:~/swefs/sefs_test1$
```

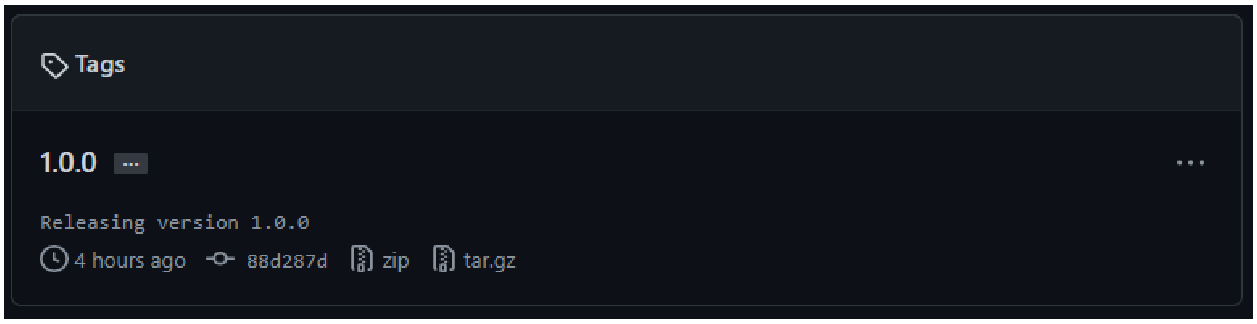
If we now go to GitHub we can see that there is one tag associated with the repo:


main
2 branches
1 tag
Go to file
Add file
Code

jtstanley Merge pull request #4 from SoftwareEngineering4Scientists/branch3 ... 55c0e6a 2 days ago 11 commits

README.md	Initial commit	4 days ago
file1.txt	I've added a third line to file1	2 days ago
file2.txt	Added second line to file2 of the repo	2 days ago
file3.txt	Added another line to file3 when I shouldn't have?	2 days ago

Here we see the contents of the tag and links to download a zipped version of the code:

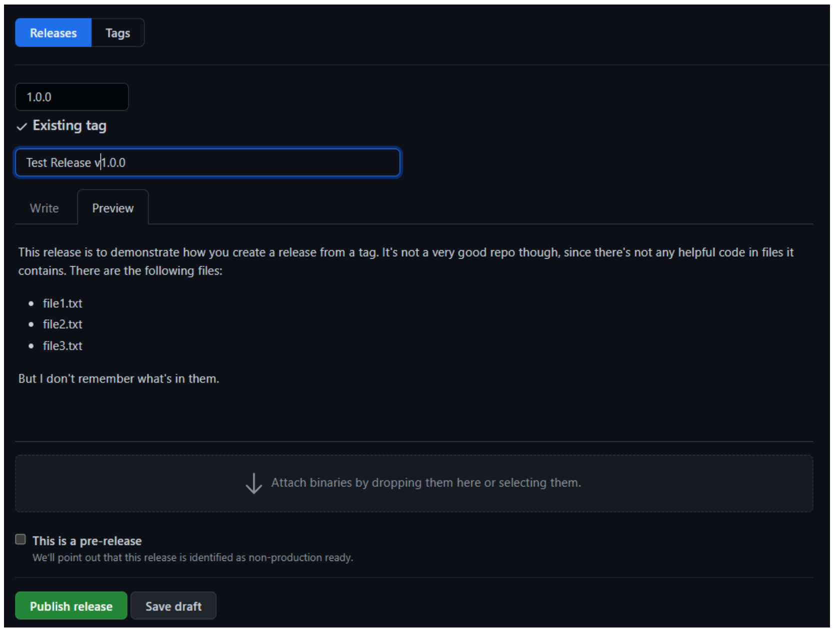

Tags

1.0.0

```
Releasing version 1.0.0
```

4 hours ago 88d287d zip tar.gz

Finally, to advertise that the particular tag is the "official release" of the software version you can select "create release" in the three dot menu at right. This will bring up a page where you can add information. You should write a thorough explainer about the release in the space provided. This should at least contain all the info you put in the tag message. Much like the repo README, this section can be markdown formatted. Once finished, click "Publish release" and you're done.


Releases Tags

1.0.0
Existing tag

Test Release v1.0.0

Write Preview

This release is to demonstrate how you create a release from a tag. It's not a very good repo though, since there's not any helpful code in files it contains. There are the following files:

- file1.txt
- file2.txt
- file3.txt

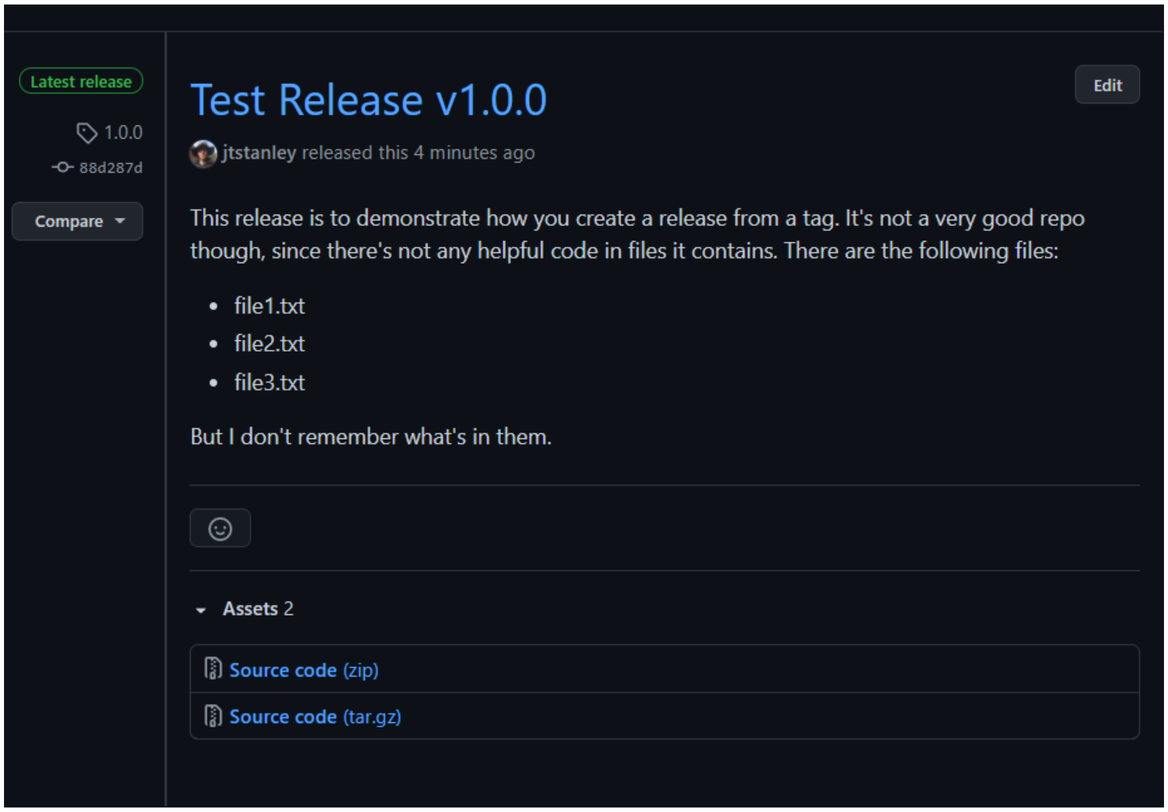
But I don't remember what's in them.

Attach binaries by dropping them here or selecting them.

This is a pre-release
We'll point out that this release is identified as non-production ready.

Publish release Save draft

You have now created a versioned, easily accessible, well-documented release of your software, available for others to use!



Test Release v1.0.0

jtstanley released this 4 minutes ago

This release is to demonstrate how you create a release from a tag. It's not a very good repo though, since there's not any helpful code in files it contains. There are the following files:

- file1.txt
- file2.txt
- file3.txt

But I don't remember what's in them.

Assets 2

Source code (zip)

Source code (tar.gz)