License Statement

About

This workshop style self-learning material is to getting you from zero knowledge about 'git' to being a hero in using it. Git is an important software configuration tool used for software versioning popularly used in open-source development. So, the target audience of this material is for any aspiring individuals (college students or engineers) who wants to acquire the skill. The material assumes that you have Linux basic shell commands and bash knowledge.

Just like any material created, there could be mistake, typo or grammar (since I am not native speaker). So, I would appreciate that you share with me so that I can continue to update it over time.

Author

Ong Boon Leong works for Intel as Principal Software Engineer under Internet of Things Group (IOTG). He is passionate about developing local talents in open-source software development and over these years have been giving adjunct lectures in local and regional educational institution and universities. He has been in the embedded software industries for close to 20 years now (since 2001) and worked on numerous different technologies ranging from IXP network processor, Windows CE/MSAuto, I/O, Audio, Low Power IA processor series, Yocto Project BSPs for Intel's Xeon/Core/Atom/ Quark product series and in recent years are leading the software development for Ethernet and Time-Sensitive Networking.

Email: boon.leong.ong@intel.com
Linked-In: https://www.linkedin.com/in/boon-leong-ong-81255b1/

Prepared by Ong Boon Leong

Linux-based development machine preparation:

- The symbol '`$`' marks the shell command prompt and symbol '`#`' indicates a comment.
- The term 'repo' refers to repository (a software configuration management system that tracks changes to source code)
- This workshop assumes that there is no network firewall or proxy between your development machine and Internet. However, most universities or enterprises deploy network firewall or proxy, so you will need to configure the network settings accordingly. **Appendix A** provides guidance on configuration required on a Linux-based Ubuntu OS.
- https://www.yoctoproject.org/docs/current/ref-manual/ref-manual.html#required-packages-for-the-build-host provides a list of essential software packages for different type of Linux distro such as Ubuntu/Debian and Fedora. Please make sure that you have installed them on your development machine. For example, Ubuntu development machine will need below softwares:

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo \
       gcc-multilib build-essential chrpath socat cpio python3 \
       python3-pip python3-pexpect xz-utils debianutils iputils-ping \
       python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev \
       pylint3 xterm
```

- To use auto-completion feature when typing a bash or git command, please install `bash-completion` package.

```
$ sudo apt-get install bash-completion
```

- It is important to configure your git before you start. An example is shown in **Appendix B**.
- An example of vim configuration is in **Appendix D**.
- We will use https://git.kernel.org/pub/scm/network/ethtool/ethtool.git for this lab-work.
- Finally, use '`git help`' to check its version is relative new (v2.25.1).

```
$ git help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone             Clone a repository into a new directory
   init              Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add               Add file contents to the index
   mv                Move or rename a file, a directory, or a symlink
   restore           Restore working tree files
   rm                Remove files from the working tree and from the index
   sparse-checkout   Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
   bisect            Use binary search to find the commit that introduced a bug
   diff              Show changes between commits, commit and working tree, etc
   grep              Print lines matching a pattern
   log               Show commit logs
   show              Show various types of objects
   status            Show the working tree status

grow, mark and tweak your common history
   branch            List, create, or delete branches
   commit            Record changes to the repository
   merge             Join two or more development histories together
   rebase            Reapply commits on top of another base tip
   reset             Reset current HEAD to the specified state
   switch            Switch branches
   tag               Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch             Download objects and refs from another repository
   pull              Fetch from and integrate with another repository or a local branch
   push              Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
```

Basic Git - 101

To git clone an open-source (remotely hosted) software over Internet to your work directory

```
$ mkdir -p ~/repos; cd ~/repos
$ git clone git://git.kernel.org/pub/scm/network/ethtool/ethtool.git
$ cd ethtool
```

Note: "master" branch is automatically checked out after 'git clone' operation has completed.
Note: If the git clone operation fails, it is likely that you are behind firewall and you need to setup your network proxy setting accordingly. Please refer to Appendix A.

To show current working branch and all available remote branch

```
# To show current working git branch
$ git branch

# To show all (working, local and remote git branches)
$ git branch -a
```

Note: The '*' besides master branch indicates that it is the currently check-out branch
Note: In Appendix A, PS1 environment variable (inside ~/.bashrc) is set to contains `$(__git_ps1 "(%s)")` to have shell prompt automatically shows git branch.

```
sashimi@glass:testl $ cd ethtool/
sashimi@glass:ethtool (master) $
```

To show the commit history/log in full or in one-liner format

```
# To show full log (both commit title and message)
# Note: to exit git log, simply type "q" and "Enter" key.
$ git log

# To show only commit title (one-liner)
$ git log --oneline

# To show commit title for a specific file
$ git log --oneline -- ethtool.c
```

Note: Navigate to older commit history using down/up key. Also, observe that each commit has unique commit ID (SHA-1 hash value).
Note: We have set git alias (inside ~/.gitconfig) for 'git log --oneline' as 'git ol', so you may also use 'git ol' as short command.
Note: The file filter search separator ' -- ' is very useful to limit the scope of the git command (include git show, git diff and other commands). So, '`git log --oneline -- ethtool.c`' is used to list one-liner git log related to the file ethtool.c.

To show a git commit (message and change)

```
$ git show 124a3c06d1c3
```

Note: Observe that commit title is one-liner and usually less than 74-character. A good commit title is crisp in explaining the intention of the commit. A good commit body explains the what, why and how about the changes. Lastly, a good commit also has clear 'signed-off-by', a.k.a. S.O.B.

Note: In "Sign your work" section in https://www.kernel.org/doc/html/v5.10/process/submitting-patches.html, there are good materials that explain how numerous tags (Signed-off-by, Tested-by, Reviewed-by) are used Linux kernel community and the practices are often adopted in other open source projects other than Linux. However, it is always good to scroll through commit history/log or read through mailing list of a software project to understand the community practices.

To make change on source code, simply use any text editor, such as vscode or vim.

```
# In this example, we use 'vim' editor to add extra text
# "This is my edit" to the end of the README file.
$ vim README

# Create a new file with one-liner 'Hello World' text.
$ echo "Hello World" > new_file.txt
```

To show status of working directory (tracked and untracked file by git)

```
$ git status
```

Note: Observe that there are both "changes not staged for commit" for tracked file (in this case README) and "untracked files" (for new_file.txt)

To show changes made on tracked file

```
$ git diff -- README
```

Note: '+' and '-' marks the lines that have addition/modification/ delete.

To restore the change of a tracked file to its previous state

```
$ git restore README
```

To add changes to git indexed staging area ('stage the change' before a commit)

```
$ git add README new_file.txt
```

To show the changes added to the staging area

```
$ git diff --cached
# To show per-file changes in staging
$ git diff --cached -- README
```

If you are not satisfied with the change, you can continue to edit the file and use 'git add' to add the changes to the staging area.

Finally, to commit (make it a change history) the staged change into repository

```
$ git commit -s
```

Note: A good "git commit" write-up: (1) one-liner, maximum 74-character width in the format 'subject: crisp title description', (2) commit body that has details explaining the why, what and how of the changes (3) the SOB (automatically added because of '`-s`').

Now that you have committed a new change, you can view the new commit as follow:

```
$ git log --oneline
$ git show
```

Let's assume that you made a mistake in README and you want to change to commit,

```
# Edit your change by using editor
$ vim README

# Then, add the new change into staging and commit the amendment
$ git add README
$ git commit --amend
```

Note: The above "vim README" step is not needed if you just want to edit the commit message.
Note: The above steps are only applicable for the most recent commit (a.k.a. HEAD commit)

Observe that the updated commit also now has a new commit ID.

```
$ git log --oneline
```

Basic Git - 102

To show the list of tracked git repositories

```
$ git remote -v
origin  git://git.kernel.org/pub/scm/network/ethtool/ethtool.git (fetch
origin  git://git.kernel.org/pub/scm/network/ethtool/ethtool.git (push)
```

Note: the 'origin' is the default name of a tracked repo and has two URLs (often the same) for git 'fetch' and 'push' operation (to be explained later).

Let's assume that you are interested to follow a fork of `ethtool` repo contributed by another developer (https://github.com/pamolloy/ethtool), you will add it to your tracked repository list.

```
$ git remote add pamo https://github.com/pamolloy/ethtool.git
```

Note: The repo name 'pamo' is just an example and can be any sensibly chosen name.
Note: In real-world, all `ethtool` (a tool used for controlling Ethernet controller) is developed on the upstream repository at https://git.kernel.org/pub/scm/network/ethtool/ethtool.git

To show the newly added tracked repo:

```
$ git remote -v
origin  git://git.kernel.org/pub/scm/network/ethtool/ethtool.git (fetch)
origin  git://git.kernel.org/pub/scm/network/ethtool/ethtool.git (push)
pamo    https://github.com/pamolloy/ethtool.git (fetch)
pamo    https://github.com/pamolloy/ethtool.git (push)
```

Note: At this stage, the git objects (commit, labels, etc) for 'pamo' repo has not been fetched (downloaded) into your local repo yet.

To synchronize a tracked repo with its upstream (remotely hosted) repo:

```
$ git fetch pamo
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (2/2), done.
remote: Total 4 (delta 2), reused 2 (delta 2), pack-reused 2
Unpacking objects: 100% (4/4), 762 bytes | 69.00 KiB/s, done.
From https://github.com/pamolloy/ethtool
 * [new branch]                master     -> pamo/master
```

Note: Observe that during git fetch operation, the difference between local 'pamo' repo and its upstream repo is calculated, compressed, downloaded, and finally unpacked. This makes your local repo to have same states as its upstream repo.
Note: If you perform another '`git fetch pamo`' command now, you will observe that there is no more new update (in fact the 'pamo' repo is actually inactive years ago as observable from its commit history in github).

To show all git branches tracked by both 'origin' and 'pamo' repositories:

```
$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/ethtool-3.4.y
  remotes/origin/master
  remotes/origin/next
  remotes/pamo/master
```

Note: Observe that 'pamo' repo only has one 'master' branch and the 'origin' repo has 3 branches (`ethtool-3.4.y`, `next` and `master`).

To check-out the 'master' branch from 'pamo' repository:

```
$ git checkout -b pamo-master --track pamo/master
$ git branch -vv
```

To show the upstream repo of local branch:

```
$ git branch -vv
* master       8a38d9228960 [origin/master] Release version 5.12.
  pamo-master 0975dca47038 [pamo/master] Merge pull request #1 from pamolloy/features-long-name
```

Note: It is important to know that 'the upstream repo' here is as update as the last git fetch operation.
Note: If you have added commits on-top of local branch and you wish to update the local branch to be the same state as the upstream branch, you will need to use '`git pull --rebase`' so that your local commits are rebased on-top of the latest commit history from the upstream branch.

To switch to another git branch, say 'master' branch:

```
$ git checkout master

# Now, let look for the new git commit that we added in 101 section
$ git log --oneline
d605b92d9a94 (HEAD -> master) ethtool: edit README and add hello world text
8a38d9228960 (tag: v5.12, origin/next, origin/master, origin/HEAD) Release version 5.12.
ed132a64b3c3 test: avoid compile warnings in test_cmdline()
c8e6dd015c4d test: mark unused arguments and function in TEST_ETHTOOL build
```

Note: The commit 'd605b92d9a94' will be different in your working directory.

Now, switch back to '`pamo-master`' branch.

```
$ git checkout pamo-master
```

To cherry-pick a commit from '`master`' to '`pamo-master`' branch:

```
$ git cherry-pick -s -x d605b92d9a94
$ git show
Author: Ong Boon Leong <[redacted]>
Date:   Wed May 5 14:07:51 2021 +0800

    ethtool: edit README and add hello world text

    Signed-off-by: Ong Boon Leong <[redacted]>
    (cherry picked from commit d605b92d9a94705d067dd14e4dc7a6355c7356fc)
    Signed-off-by: Ong Boon Leong <[redacted]>

diff --git a/README b/README
index 9e0320553885..6ec9ac9a40bd 100644
--- a/README
+++ b/README
@@ -1,2 +1,4 @@
 ethtool is a small utility for examining and tuning your ethernet-based
 network interface.  See the man page for more details.
+
+This is my edit.
diff --git a/new_file.txt b/new_file.txt
new file mode 100644
index 000000000000..557db03de997
--- /dev/null
+++ b/new_file.txt
@@ -0,0 +1 @@
+Hello World
```

Note: The option '-s' means automatically adds a signed-off-by tag and '-x' means adding cherry-pick origin commit ID before the signed-off-by. The use '-x' is very important if you are cherry-picking from public open-source repo so that other developers know the origin of the changes. It is often used as a record to for your fellow developer to cherry-pick the same commit into their own repo. '-x' is of no use (in fact does not have any meaning) if you are not using 'upstream' repo. This is because the commit ID (in cherry picked …tag) is only unique to your local branch and not traceable to anyone else.

Now, let's add two more changes/commits to 'pamo-master'.

```
# Make the 2nd change according to below 'git show'.
$ vi README
$ git add README
$ git commit -s -m "README: my 2nd edit"
$ git show
commit 7dae1df23115e3a3867324bdc13475a590301c76 (HEAD -> pamo-master)
Author: Ong Boon Leong <[redacted]>
Date:   Wed May 5 14:36:08 2021 +0800

    README: my 2nd edit

    Signed-off-by: Ong Boon Leong <[redacted]>

diff --git a/README b/README
index 6ec9ac9a40bd..4c53379e961a 100644
--- a/README
+++ b/README
@@ -1,4 +1,5 @@
 ethtool is a small utility for examining and tuning your ethernet-based
 network interface.  See the man page for more details.

-This is my edit.
+This is my 2nd edit.
+This is my edit. I also changed here.


# Make the 3rd change according to below 'git show'
$ vi README
$ git add README
$ git commit -s -m "README: my 3rd edit"
$ git show
commit 385e8d87591a446da788af5d9568934591730e30 (HEAD -> pamo-master)
Author: Ong Boon Leong <[redacted]>
Date:   Wed May 5 14:39:30 2021 +0800

    README: my 3rd edit

    Signed-off-by: Ong Boon Leong <[redacted]>

diff --git a/README b/README
index 4c53379e961a..6e375c18ef78 100644
--- a/README
+++ b/README
@@ -1,5 +1,5 @@
 ethtool is a small utility for examining and tuning your ethernet-based
 network interface.  See the man page for more details.

-This is my 2nd edit.
-This is my edit. I also changed here.
+This is my 3rd edit.
+This is my edit. Now, I changed here again.
```

Note: The option '-m' allows us to input the one-liner git title and this is often used for git commit that is obvious and does not need lengthy git commit body (description). In another words, if you have a multi-line commit message, please use '`git commit -s`' instead.

To show the git commit history:

```
$ git log --oneline
385e8d87591a (HEAD -> pamo-master) README: my 3rd edit
7dae1df23115 README: my 2nd edit
8687e433c2bd ethtool: edit README and add hello world text
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
```

Next, we want to create a new branch and to try out '`git rebase`' command later.

```
$ git branch pamo-master-test
$ git checkout pamo-master-test
$ git log --oneline
385e8d87591a (HEAD -> pamo-master-test, pamo-master) README: my 3rd edit
7dae1df23115 README: my 2nd edit
8687e433c2bd ethtool: edit README and add hello world text
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name


$ git branch -vv
  master           d605b92d9a94 [origin/master: ahead 1] ethtool: edit README and add hello world text
  pamo-master      385e8d87591a [pamo/master: ahead 3] README: my 3rd edit
* pamo-master-test 385e8d87591a README: my 3rd edit
```

Note: The '`pamo-master-test`' branch does not have upstream branch. It is simply a new branch that at this point has the same commit history as '`pamo-master`' branch.

Note: In git, 'pamo-master' and 'pamo-master-test' branches use their respective HEAD to point to commit. Since a commit has parent commit and the relationship goes on up the lineage of git history, any change to a particular commit along the lineage will mean the entire commit history now has different commit ID. As a matter of fact, any change to a git commit (code change, authorship, date/time, parent of the commit and other attributes) will produce a new SHA-1 hash computation that is essentially the commit ID.

Let's say that we want to discard the 3rd edit on README from the history tracked 'pamo-master-test' branch.

```
# The commit-ID before 3rd README edit is 7dae1df23115.
$ git reset --hard 7dae1df23115
$ git log --oneline
7dae1df23115 (HEAD -> pamo-master-test) README: my 2nd edit
8687e433c2bd ethtool: edit README and add hello world text
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
```

Note: The commit '`7dae1df23115`' for the 3rd README commit will be different in your working git repo. So, you will need to use the correct commit ID in your case.

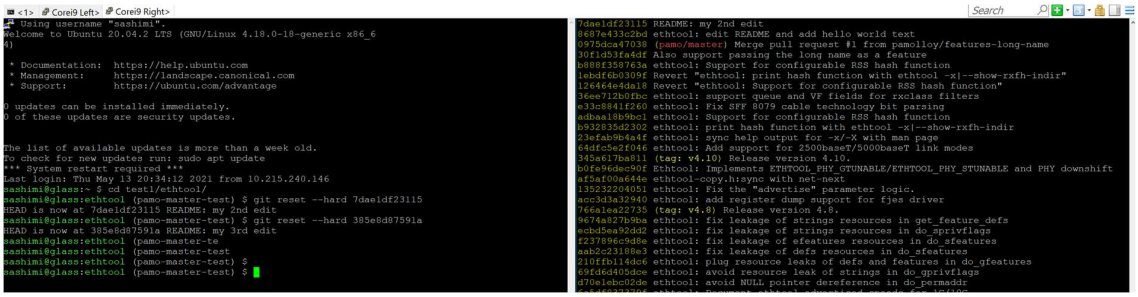
Now, you regret about removing the 3rd README edit commit and you want to reverse the removal.

```
# The commit-ID before 3rd README edit is 385e8d87591a.
$ git reset --hard 385e8d87591a
$ git log --oneline
385e8d87591a (HEAD -> pamo-master-test, pamo-master) README: my 3rd edit
7dae1df23115 README: my 2nd edit
8687e433c2bd ethtool: edit README and add hello world text
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
```

Note: By now, you should see the nature of git branch 'pamo-master-test' here. The HEAD of '`pamo-master-test`' branch is really a pointer and the '`git reset --hard`' command can

be used to reset the HEAD to point at a specific commit ID in a git repo (this includes commit that is also tracked by another git branch).

Tip: If you are worried of messing up your git history, you could always just make a copy of "`git log --oneline`" before you perform any "`git reset --hard <other commit>`" or simply just open two terminals side-by-side, the right terminal is '`git log --oneline`' which allows you to scroll up and down to obtain the desired commit ID. In addition, until you exit the "`git log --oneline`" by entering '`q`' + '`Enter`', the git history does not change even though you have done multiple "`git reset --hard`" on left terminal on the same git repo.



Advanced Git - 201

We have learned about '`git commit --amend`' in Basic Git 101 and this command is used to edit the HEAD commit (git commit write-up or code changes). In this section, we will learn about '`git rebase -i <commit-ID>`' that is used to reapply commits (a change to make commit change) on top of a commit base.

Let's show the commit history on '`pamo-master-test`' branch:

```
$ git log --oneline
385e8d87591a (HEAD -> pamo-master-test, pamo-master) README: my 3rd edit
7dae1df23115 README: my 2nd edit
8687e433c2bd ethtool: edit README and add hello world text
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-nam
30f1d53fa4df Also support passing the long name as a feature
```

To change commits including and on-top of a commit base, say "`8687e433c2bd ethtool: edit README and add hello world text`":

```
$ git rebase -i 8687e433c2bd^
pick 8687e433c2bd ethtool: edit README and add hello world text
pick 7dae1df23115 README: my 2nd edit
pick 385e8d87591a README: my 3rd edit

# Rebase 0975dca47038..385e8d87591a onto 0975dca47038 (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

Note: The character '^' is used here so that the commit '`8687e433c2bd`' is included in the commits apply list.

Note: The order of the commits apply order is such that the most recent commit at the bottom of the list. It is completely in reverse order in comparison to the list by '`git log --oneline`'.

Note: The commit apply list shows all the commits from the base (in this case `8687e433c2bd` till the HEAD). Each of the commit is default to "`pick`" command which, as its name suggests, picks the commit for apply. There are other commands such as `reword`, `edit`, `squash`, etc. Instead of typing the whole command name, you can just use the character, e.g. for 'rewording git commit message', use 'r' or 'reword'.

A) git rebase: "reword" command

To reword commit message of '`8687e433c2bd ethtool: edit README and add hello world text`', change '`pick`' to '`r`'. Then, save the commit apply list (for vim, press "`ESC`" then "`:wq`").

```
r 8687e433c2bd ethtool: edit README and add hello world text
pick 7dae1df23115 README: my 2nd edit
pick 385e8d87591a README: my 3rd edit
```

Then, the terminal will use '`vim`' to open the git commit message for "`ethtool: edit README and add hello world text`" as shown below. So, make the change for the commit message as shown below.

Original commit message:

```
ethtool: edit README and add hello world text

Signed-off-by: Ong Boon Leong <[redacted]>
(cherry picked from commit d605b92d9a94705d067dd14e4dc7a6355c7356fc)
Signed-off-by: Ong Boon Leong <[redacted]>

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Wed May 5 14:07:51 2021 +0800
#
# interactive rebase in progress; onto 0975dca47038
# Last command done (1 command done):
#    r 8687e433c2bd ethtool: edit README and add hello world text
# Next commands to do (2 remaining commands):
#    pick 7dae1df23115 README: my 2nd edit
#    pick 385e8d87591a README: my 3rd edit
# You are currently editing a commit while rebasing branch 'pamo-master-test' on '0975dca47038'.
#
# Changes to be committed:
#>------modified:   README
#>------new file:   new_file.txt
#
```

New commit message:

```
ethtool: 1st README change and add hello world text

I just change the git title above a bit.

Signed-off-by: Ong Boon Leong <[redacted]>
(cherry picked from commit d605b92d9a94705d067dd14e4dc7a6355c7356fc)
Signed-off-by: Ong Boon Leong <[redacted]>
```

After the commit message has been saved, the rest of the git commits will be applied according to the commits apply list.

```
sashimi@glass:ethtool (pamo-master-test) $ git rebase -i 8687e433c2bd^
[detached HEAD 990d1432a4bb] ethtool: 1st README change and add hello world text
 Date: Wed May 5 14:07:51 2021 +0800
 2 files changed, 3 insertions(+)
 create mode 100644 new_file.txt
Successfully rebased and updated refs/heads/pamo-master-test.
```

Show the git history log, then take note that the commit ID for all the 3 commits have changed.

```
$ git log --oneline
```

Note: What you have just performed is to use git rebase to change git commit message that is not HEAD. Please remember that if you intend to change the git commit message for HEAD commit, you can simply just use '`git commit --amend`'.

```
5be9fe3b3bb9 (HEAD -> pamo-master-test) README: my 3rd edit
0be4990b933b README: my 2nd edit
990d1432a4bb ethtool: 1st README change and add hello world text
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
```

Now, let's look at the new git commit that we have just changed before:

```
$ git show 990d1432a4bb
```

```
sashimi@glass:ethtool (pamo-master-test) $ git show 990d1432a4bb
commit 990d1432a4bb0fed97b995ef6c7ca18bba0b9738
Author: Ong Boon Leong <[redacted]>
Date:   Wed May 5 14:07:51 2021 +0800

    ethtool: 1st README change and add hello world text

    I just change the git title above a bit.

    Signed-off-by: Ong Boon Leong <[redacted]>
    (cherry picked from commit d605b92d9a94705d067dd14e4dc7a6355c7356fc)
    Signed-off-by: Ong Boon Leong <[redacted]>

diff --git a/README b/README
index 9e0320553885..6ec9ac9a40bd 100644
--- a/README
+++ b/README
@@ -1,2 +1,4 @@
 ethtool is a small utility for examining and tuning your ethernet-based
 network interface.  See the man page for more details.
+
+This is my edit.
diff --git a/new_file.txt b/new_file.txt
new file mode 100644
index 000000000000..557db03de997
--- /dev/null
+++ b/new_file.txt
@@ -0,0 +1 @@
+Hello World
```

Note: Observe that this commit is not done in accordance to software configuration good practices whereby the change of a commit should have natural scope (in this case it is a mixed bag of `README` and `new_file.txt`). So, in the next section, we will use '`edit`' command to fix it.

B) git rebase – "edit" command

To edit a (non-HEAD) commit, we start with git rebase command:

```
$ git rebase -i 990d1432a4bb^
# replace 'pick' to 'e' or 'edit' and save the commit apply list
e 990d1432a4bb ethtool: 1st README change and add hello world text
pick 0be4990b933b README: my 2nd edit
pick 5be9fe3b3bb9 README: my 3rd edit

# Rebase 0975dca47038..5be9fe3b3bb9 onto 0975dca47038 (3 commands)

# When the git rebase started, you will be presented at the
# "ethtool: 1st README change and add hello world text" commit.
# To show the commit now,
$ git show
```

```
sashimi@glass:ethtool (pamo-master-test|REBASE-i 1/3) $ git show
commit 990d1432a4bb0fed97b995ef6c7ca18bba0b9738 (HEAD)
Author: Ong Boon Leong <[redacted]>
Date:   Wed May 5 14:07:51 2021 +0800

    ethtool: 1st README change and add hello world text

    I just change the git title above a bit.

    Signed-off-by: Ong Boon Leong <[redacted]>
    (cherry picked from commit d605b92d9a94705d067dd14e4dc7a6355c7356fc)
    Signed-off-by: Ong Boon Leong <[redacted]>

diff --git a/README b/README
index 9e0320553885..6ec9ac9a40bd 100644
--- a/README
+++ b/README
@@ -1,2 +1,4 @@
 ethtool is a small utility for examining and tuning your ethernet-based
 network interface.  See the man page for more details.
+
+This is my edit.
diff --git a/new_file.txt b/new_file.txt
new file mode 100644
index 000000000000..557db03de997
--- /dev/null
+++ b/new_file.txt
@@ -0,0 +1 @@
+Hello World
```

Note: The terminal prompt now shows "`REBASE-i 1/3`" which means it is operating at the 1st of the 3 commits of the apply list.

Let's just say we want to make two changes (`README` and `new_file.txt`). Recognize that the commit is also at new HEAD, we can use "`git reset HEAD~`" to the previous state before the change was committed.

```
$ git reset HEAD~
# At this point, you are at the point before the commit was performed.
$ git status
```

```
sashimi@glass:ethtool (pamo-master-test|REBASE-i 1/3) $ git status
interactive rebase in progress; onto 0975dca47038
Last command done (1 command done):
   e 990d1432a4bb ethtool: 1st README change and add hello world text
Next commands to do (2 remaining commands):
   pick 0be4990b933b README: my 2nd edit
   pick 5be9fe3b3bb9 README: my 3rd edit
  (use "git rebase --edit-todo" to view and edit)
You are currently splitting a commit while rebasing branch 'pamo-master-test' on '0975dca47038'.
  (Once your working directory is clean, run "git rebase --continue")

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:   README

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        new_file.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Tip: if you want to save some time from re-writing a commit message, simply open two terminals with one of the terminal has 'git show' command used to show the git commit message.

Next, we add and commit the change for README.

```
$ git add README
$ git commit -s
# If we just paste the previous commit message as part of the
# new commit message, we need to be aware that the cherry picked
# tag (from -x) earlier is no longer valid. Besides, there are multiple
# SOBs in the commit message that is not relevant.
```

```
ethtool: 1st README change and add hello world text

I just change the git title above a bit.

Signed-off-by: Ong Boon Leong <[redacted]>
(cherry picked from commit d605b92d9a94705d067dd14e4dc7a6355c7356fc)
Signed-off-by: Ong Boon Leong <[redacted]>

Signed-off-by: Ong Boon Leong <[redacted]>
```

```
# So, we will change commit message as follow and save the commit:
```

```
ethtool: 1st README change

I just change the git title above a bit.

Signed-off-by: Ong Boon Leong <[redacted]>

# Please enter the commit message for your changes. Lines starting
```

```
# Now, we have gotten the first part of what we wanted to achieve.
# i.e. to split the original commit (990d1432a4bb) into one commit for
# the README change
$ git show
```

```
sashimi@glass:ethtool (pamo-master-test|REBASE-i 1/3) $ git show
commit c4cdb86c0db1fe87c0a9c3bb357d4a0ae0cdc1ab (HEAD)
Author: Ong Boon Leong <[redacted]>
Date:   Fri May 14 11:21:48 2021 +0800

    ethtool: 1st README change

    I just change the git title above a bit.

    Signed-off-by: Ong Boon Leong <[redacted]>

diff --git a/README b/README
index 9e0320553885..6ec9ac9a40bd 100644
--- a/README
+++ b/README
@@ -1,2 +1,4 @@
 ethtool is a small utility for examining and tuning your ethernet-based
 network interface.  See the man page for more details.
+
+This is my edit.
```

Next, we add another commit for '`new_file.txt`'.

```
$ git status
```

```
sashimi@glass:ethtool (pamo-master-test|REBASE-i 1/3) $ git status
interactive rebase in progress; onto 0975dca47038
Last command done (1 command done):
   e 990d1432a4bb ethtool: 1st README change and add hello world text
Next commands to do (2 remaining commands):
   pick 0be4990b933b README: my 2nd edit
   pick 5be9fe3b3bb9 README: my 3rd edit
  (use "git rebase --edit-todo" to view and edit)
You are currently editing a commit while rebasing branch 'pamo-master-test' on '0975dca47038'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        new_file.txt

nothing added to commit but untracked files present (use "git add" to track)
```

```
$ git add new_file.txt
$ git commit -s
# Add the commit message as below and save it.
```

```
new_file: initial version of the new_file that says hello world

Signed-off-by: Ong Boon Leong <[redacted]>

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# interactive rebase in progress; onto 0975dca47038
# Last command done (1 command done):
#    e 990d1432a4bb ethtool: 1st README change and add hello world text
# Next commands to do (2 remaining commands):
#    pick 0be4990b933b README: my 2nd edit
#    pick 5be9fe3b3bb9 README: my 3rd edit
# You are currently splitting a commit while rebasing branch 'pamo-master-test' on '0975dca47038'.
#
# Changes to be committed:
#>------new file:   new_file.txt
```

```
# Next, we show the commit history again. There are our two new commits
$ git log --oneline
```

```
b5968d7cc363 (HEAD) new_file: initial version of the new_file that says hello world
c4cdb86c0db1 ethtool: 1st README change
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
30f1d53fa4df Also support passing the long name as a feature
```

```
# At this point, if we want to add another new file or make
# another commit we can do the usual 'edit + git add + git commit'
# sequence that we have been using a lot by now
$ echo 'Good Morning to Alexa' > alexa.txt
$ git add alexa.txt
$ git commit -s -m "alexa: Good morning message"
[detached HEAD b18eb8037fe4] alexa: Good morning message
 1 file changed, 1 insertion(+)
 create mode 100644 alexa.txt
sashimi@glass:ethtool (pamo-master-test|REBASE-i 1/3) $


# Take note that we are still at "REBASE-i 1/3" and to continue
# applying the rest of commit list
$ git rebase -continue
sashimi@glass:ethtool (pamo-master-test|REBASE-i 1/3) $ git rebase --continue
Successfully rebased and updated refs/heads/pamo-master-test.
sashimi@glass:ethtool (pamo-master-test) $
```

C) git rebase – "squash" command and commits reordering with "pick"

Let's look at the commit list that we want to operate on again:

```
$ git log --oneline
ac169ddb11e4 (HEAD -> pamo-master-test) README: my 3rd edit
2177b8aef9b5 README: my 2nd edit
b18eb8037fe4 alexa: Good morning message
b5968d7cc363 new_file: initial version of the new_file that says hello world
c4cdb86c0db1 ethtool: 1st README change
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
```

Let's say we want to move the order of '`c4cdb86c0db1 ethtool: 1st README change`' to be after '`b18eb8037fe4 alexa: Good morning message`'. We can use the '`git rebase`' apply commit list again.

```
$ git rebase -i c4cdb86c0db1^
# The original commit apply list
pick c4cdb86c0db1 ethtool: 1st README change
pick b5968d7cc363 new_file: initial version of the new_file that says hello world
pick b18eb8037fe4 alexa: Good morning message
pick 2177b8aef9b5 README: my 2nd edit
pick ac169ddb11e4 README: my 3rd edit

# Rebase 0975dca47038..ac169ddb11e4 onto 2177b8aef9b5 (5 commands)


# The new commit apply list
# Note: For 'vim' editor, under command mode (press ESC):
# to cut a line, type 'cc' then move your cursor to
# 'alexa: Good morning message', then type 'p' to paste a cut line.
# To delete the previous cut line, type 'dd'.
# To save the apply list, press ":wq"
pick b5968d7cc363 new_file: initial version of the new_file that says hello world
pick b18eb8037fe4 alexa: Good morning message
pick c4cdb86c0db1 ethtool: 1st README change
pick 2177b8aef9b5 README: my 2nd edit
pick ac169ddb11e4 README: my 3rd edit


# Show the new git history again
$ git log --oneline
d24c9be96405 (HEAD -> pamo-master-test) README: my 3rd edit
c06f4d74e5d4 README: my 2nd edit
f23e892efbbe ethtool: 1st README change
cb3fe007855a alexa: Good morning message
ec46a7ba1865 new_file: initial version of the new_file that says hello world
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
```

Let's continue to squash (combine) commits related to README:

```
$ git rebase -i f23e892efbbe^
# The original commit apply list
pick f23e892efbbe ethtool: 1st README change
pick c06f4d74e5d4 README: my 2nd edit
pick d24c9be96405 README: my 3rd edit

# Rebase cb3fe007855a..d24c9be96405 onto cb3fe007855a (3 commands)


# The new commit apply list, to combine both 2nd and 3rd commit
# README change
pick f23e892efbbe ethtool: 1st README change
s c06f4d74e5d4 README: my 2nd edit
s d24c9be96405 README: my 3rd edit

# Rebase cb3fe007855a..d24c9be96405 onto cb3fe007855a (3 commands)


# When we are squashing git commits, we will be presented with commit
# message editing automatically that combines older commit messages.
# Note there are 3 SOBs and you really just need one SOB for the new
# commit.
# This is a combination of 3 commits.
# This is the 1st commit message:

ethtool: 1st README change

I just change the git title above a bit.

Signed-off-by: Ong Boon Leong <[redacted]>

# This is the commit message #2:

README: my 2nd edit

Signed-off-by: Ong Boon Leong <[redacted]>

# This is the commit message #3:

README: my 3rd edit

Signed-off-by: Ong Boon Leong <[redacted]>

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Fri May 14 11:21:48 2021 +0800
#
# interactive rebase in progress; onto cb3fe007855a
# Last commands done (3 commands done):
#    s c06f4d74e5d4 README: my 2nd edit
#    s d24c9be96405 README: my 3rd edit
# No commands remaining.
# You are currently rebasing branch 'pamo-master-test' on 'cb3fe007855a'.
#
# Changes to be committed:
#>------modified:   README
#
```

```
# Let's just edit commit message as follow:
# This is a combination of 3 commits.
# This is the 1st commit message:
README: Add additional information.

Signed-off-by: Ong Boon Leong <[redacted]>

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Fri May 14 11:21:48 2021 +0800
#
# interactive rebase in progress; onto cb3fe007855a
# Last commands done (3 commands done):
#    s c06f4d74e5d4 README: my 2nd edit
#    s d24c9be96405 README: my 3rd edit
# No commands remaining.
# You are currently rebasing branch 'pamo-master-test' on 'cb3fe007855a'
#
# Changes to be committed:
#>------modified:   README
#


# Observe that commit history now only has one commit for README
# instead 3 commits earlier.
8f1a50dd1f0a (HEAD -> pamo-master-test) README: Add additional information.
cb3fe007855a alexa: Good morning message
ec46a7ba1865 new_file: initial version of the new_file that says hello world
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name


# Show the commit for README now
$ git show
commit 8f1a50dd1f0a2f24092153e52107fad1d66b4171 (HEAD -> pamo-master-test)
Author: Ong Boon Leong <[redacted]>
Date:   Fri May 14 11:21:48 2021 +0800

    README: Add additional information.

    Signed-off-by: Ong Boon Leong <[redacted]>

diff --git a/README b/README
index 9e0320553885..6e375c18ef78 100644
--- a/README
+++ b/README
@@ -1,2 +1,5 @@
 ethtool is a small utility for examining and tuning your ethernet-based
 network interface.  See the man page for more details.
+
+This is my 3rd edit.
+This is my edit. Now, I changed here again.
```

As summary, we have use '`squash`' and '`pick`' command in git rebase to combine commits and rearrange the order of commits. This technique is important because as you are developing/debugging software, often your working commits are not in the right order to be sent upstream. So, it is important to rearrange and clean-up your commits before they are shared with upstream communities so that they appear in logical sense.

Advanced Git - 202

We have seen in Advanced Git 201 earlier that the '`git rebase`' allows us to reapply a series of git commit list (with changes, commit reordering, squashed commits) on-top of a base. In this section, we will learn about aborting a git rebase command or resolving a conflict that happen when applying commits.

Let's look at the commit list that we want to operate on and now we use '`ec46a7ba1865 new_file: initial version of the new_file that says hello world`' as the commit base:

```
$ git log -oneline
8f1a50dd1f0a (HEAD -> pamo-master-test) README: Add additional information.
cb3fe007855a alexa: Good morning message
ec46a7ba1865 new_file: initial version of the new_file that says hello world
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name


$ git rebase -i ec46a7ba1865^
# Let's use "edit" to change the 'ec46a7ba1865' commit
e ec46a7ba1865 new_file: initial version of the new_file that says hello world
pick cb3fe007855a alexa: Good morning message
pick 8f1a50dd1f0a README: Add additional information.

# Rebase 0975dca47038..8f1a50dd1f0a onto 0975dca47038 (3 commands)


# Then, change the new_file.txt as follow
Alice says, "Hello World!".


# View the difference (change) that you just made
$ git diff
sashimi@glass:ethtool (pamo-master-test|REBASE-i 1/3) $ git diff
diff --git a/new_file.txt b/new_file.txt
index 557db03de997..7d7b47ac0a7f 100644
--- a/new_file.txt
+++ b/new_file.txt
@@ -1 +1,2 @@
-Hello World
+Alice says, "Hello World!".


# Now, add and commit the change.
$ git add new_file.txt
$ git commit --amend
# Also, edit the git commit message as follow:
new_file: initial version of the new_file that says hello world

Now, Alice says Hello World.

Signed-off-by: Ong Boon Leong <[redacted]>
```

```
# Show the git commit at REBASE-i 1/3
$ git show
```

```
sashimi@glass:ethtool (pamo-master-test|REBASE-i 1/3) $ git show
commit 7a8985e1f741febe3dd10b551565fa65e912370a (HEAD)
Author: Ong Boon Leong <[redacted]>
Date:   Fri May 14 11:41:37 2021 +0800

    new_file: initial version of the new_file that says hello world

    Now, Alice says Hello World.

    Signed-off-by: Ong Boon Leong <[redacted]>

diff --git a/new_file.txt b/new_file.txt
new file mode 100644
index 000000000000..7d7b47ac0a7f
--- /dev/null
+++ b/new_file.txt
@@ -0,0 +1,2 @@
+Alice says, "Hello World!".
+
```

```
# At this point, git commit history has been changed.
$ git log --oneline
```

```
7a8985e1f741 (HEAD) new_file: initial version of the new_file that says hello world
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
```

Now that we are done with REBASE-i 1/3 commit, normally we will use "`git rebase --continue`" to apply the rest of the commits. However, let's say we just want to abort the whole git rebase operation and revert to previous git commit history.

```
# To abort a git rebase operation
$ git rebase --abort
```

```
sashimi@glass:ethtool (pamo-master-test|REBASE-i 1/3) $ git rebase --abort
sashimi@glass:ethtool (pamo-master-test) $ git log --oneline
```

```
# Observe that commit history and their IDs have not changed
$ git log --oneline
```

```
8f1a50dd1f0a (HEAD -> pamo-master-test) README: Add additional information.
cb3fe007855a alexa: Good morning message
ec46a7ba1865 new_file: initial version of the new_file that says hello world
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
```

Next, we will go through an example of git conflict and resolve it. A conflict happens when the git tool does not know how to merge the commit change (added/deleted).

```
$ git rebase -i ec46a7ba1865^

# We set the apply list to "edit cb3fe007855a alexa: Good morning
message" and save the apply list
```

```
pick ec46a7ba1865 new_file: initial version of the new_file that says hello world
e cb3fe007855a alexa: Good morning message
pick 8f1a50dd1f0a README: Add additional information.

# Rebase 0975dca47038..8f1a50dd1f0a onto 0975dca47038 (3 commands)
```

```
# Take note that we are at REBASE-i 2/3 commit just before the commit
# "README: Add additional". Now, let's add some extra text to README
# as shown below
```

```
$ vi README
ethtool is a small utility for examining and tuning your ethernet-based
network interface.  See the man page for more details. Also, this
new line is added by us to show that we want to create conflict.

In addition, there is extra paragraph added here.

Besides, we have additional line added here.
```

```
# Next, git add and commit the change above
$ git add README
$ git commit -s -m "README: extra changes that causes conflict next"
$ git log --oneline
87cda0169d70 (HEAD) README: extra changes that causes conflict next
cb3fe007855a alexa: Good morning message
ec46a7ba1865 new_file: initial version of the new_file that says hello world
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
```

```
# Now, let's continue with git rebase and we got a merge conflict!!
$ git rebase --continue
sashimi@glass:ethtool (pamo-master-test|REBASE-i 2/3) $ git rebase --continue
Auto-merging README
CONFLICT (content): Merge conflict in README
error: could not apply 8f1a50dd1f0a... README: Add additional information.
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply 8f1a50dd1f0a... README: Add additional information.
sashimi@glass:ethtool (pamo-master-test|REBASE-i 3/3) $
```

To show merge conflict:

```
# To list file that has merge conflict
$ git diff --stat
```

```
sashimi@glass:ethtool (pamo-master-test|REBASE-i 3/3) $ git diff --stat
 README | Unmerged
 README | 7 +++++++
 1 file changed, 7 insertions(+)
```

```
# To show the merge conflict
$ git diff
```

```
sashimi@glass:ethtool (pamo-master-test|REBASE-i 3/3) $ git diff
diff --cc README
index 15696aab0831,6e375c18ef78..000000000000
--- a/README
+++ b/README
@@@ -1,8 -1,5 +1,15 @@@
  ethtool is a small utility for examining and tuning your ethernet-based
++<<<<<<< HEAD
 +network interface.  See the man page for more details. Also, this
 +new line is added by us to show that we want to create conflict.
 +
 +In addition, there is extra paragraph added here.
 +
 +Besides, we have additional line added here.
 +
++=======
+ network interface.  See the man page for more details.
+
+ This is my 3rd edit.
+ This is my edit. Now, I changed here again.
++>>>>>>> 8f1a50dd1f0a... README: Add additional information.
```

Note: The delimiters '<<<<<<<' and '=======' shows the content in HEAD and the delimiters '=======' and '>>>>>>>' shows the content of the commit that is supposed to be applied (in this case REBASE-i 3/3 commit).

Now, let's resolve the conflict (pick the commit in HEAD, or pick the commit in the commit 3/3 or combine them) by editing the file that has merge conflict.

```
# Edit the README as shown below.
$ vi README
```

```
ethtool is a small utility for examining and tuning your ethernet-based
network interface.  See the man page for more details.

Also, this new line is added by us to show that we want to create conflict.
In addition, there is extra paragraph added here.
Besides, we have additional line added here.

This is my edit. Now, I changed here again.
```

```
# It is always good to check what we have resolve so far
$ git diff
```

```
sashimi@glass:ethtool (pamo-master-test|REBASE-i 3/3) $ git diff
diff --cc README
index 15696aab0831,6e375c18ef78..000000000000
--- a/README
+++ b/README
@@@ -1,8 -1,5 +1,8 @@@
  ethtool is a small utility for examining and tuning your ethernet-based
- network interface.  See the man page for more details. Also, this
- new line is added by us to show that we want to create conflict.
+ network interface.  See the man page for more details.

 -This is my 3rd edit.
++Also, this new line is added by us to show that we want to create conflict.
 +In addition, there is extra paragraph added here.
-
 +Besides, we have additional line added here.
 +
+ This is my edit. Now, I changed here again.
```

```
# Now, since we no longer have any conflict as marked by the three
# delimiters ('<<<<<<<', '=======' and '>>>>>>>') earlier. If you are
# not sure what is the next step, you can use git status to cue you.
$ git status
```

```
sashimi@glass:ethtool (pamo-master-test|REBASE-i 3/3) $ git status
interactive rebase in progress; onto 0975dca47038
Last commands done (3 commands done):
   e cb3fe007855a alexa: Good morning message
   pick 8f1a50dd1f0a README: Add additional information.
  (see more in file .git/rebase-merge/done)
No commands remaining.
You are currently rebasing branch 'pamo-master-test' on '0975dca47038'.
  (fix conflicts and then run "git rebase --continue")
  (use "git rebase --skip" to skip this patch)
  (use "git rebase --abort" to check out the original branch)

Unmerged paths:
  (use "git restore --staged <file>..." to unstage)
  (use "git add <file>..." to mark resolution)
        both modified:   README

no changes added to commit (use "git add" and/or "git commit -a")
```

```
# To mark the conflict in README file has been resolved
$ git add README

# To check if there is any more conflict is left out, use either one
# of below command
$ git diff
$ git diff --stat
```

```
sashimi@glass:ethtool (pamo-master-test|REBASE-i 3/3) $ git diff
sashimi@glass:ethtool (pamo-master-test|REBASE-i 3/3) $ git diff --stat
```

```
# Finally, continue with the git rebase operation
$ git rebase -continue

# Then, you will be presented to edit the 3/3 commit message. In this
# case, we just simply save the commit message as it is.
```

```
README: Add additional information.

Signed-off-by: Ong Boon Leong <[redacted]>

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
```

```
# Let's list the commit history again
$ git log --oneline
```

```
1faeb06eb305 (HEAD -> pamo-master-test) README: Add additional information.
87cda0169d70 README: extra changes that causes conflict next
cb3fe007855a alexa: Good morning message
ec46a7ba1865 new_file: initial version of the new_file that says hello world
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
```

We have just learned how to abort a git rebase operation and how to resolve conflict. If you feel intimidated by resolving a complex conflict, the rule of thumbs is to do the change in new git branch and try to resolve the conflicts incremental fashion. The purpose of resolving conflict is to make the the state of your working git branch to a desired state. Finally, check the change between two branches are the same or as desired:

```
# List git branch
$ git branch
```

```
sashimi@glass:ethtool (pamo-master-test) $ git branch
  master
  pamo-master
* pamo-master-test
```

```
# To see the difference between git branches
$ git diff pamo-master pamo-master-test
```

```
sashimi@glass:ethtool (pamo-master-test) $ git diff pamo-master pamo-master-test
diff --git a/README b/README
index 6e375c18ef78..3838ed4b2f5e 100644
--- a/README
+++ b/README
@@ -1,5 +1,8 @@
 ethtool is a small utility for examining and tuning your ethernet-based
 network interface.  See the man page for more details.

-This is my 3rd edit.
+Also, this new line is added by us to show that we want to create conflict.
+In addition, there is extra paragraph added here.
+Besides, we have additional line added here.
+
 This is my edit. Now, I changed here again.
diff --git a/alexa.txt b/alexa.txt
new file mode 100644
index 000000000000..e2b0f593e2b8
--- /dev/null
+++ b/alexa.txt
@@ -0,0 +1 @@
+Good Morning to Alexa
```

```
# The git history for "pamo-master" and "pamo-master-test"
$ git checkout pamo-master
$ git log --oneline
```

```
385e8d87591a (HEAD -> pamo-master) README: my 3rd edit
7dae1df23115 README: my 2nd edit
8687e433c2bd ethtool: edit README and add hello world text
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
```

```
$ git checkout pamo-master-test
```

```
$ git log --oneline
1faeb06eb305 (HEAD -> pamo-master-test) README: Add additional information.
87cda0169d70 README: extra changes that causes conflict next
cb3fe007855a alexa: Good morning message
ec46a7ba1865 new_file: initial version of the new_file that says hello world
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
```

Note: Comparing the difference between git branches is good way to cross check the changes that you have made on the "pamo-master-test" branch. If you are not happy with it, you can continue to use the "git rebase" operation to change the source code or commit message until you are satisfied with it.

Advanced Git - 203

Now that you have acquired the skill to make commit change, it is time to learn about how share your commits with other developers. If you are the maintainer or owner to a public repository hosted on Internet such as under www.github.com, you can push your local git branch to the remote repository to publish your changes to the project follower. If you are a fellow developer of a project, you send a pull request or send patch-series via email.

Note: before you proceed, you need to have a GitHub account so that you can practice the steps. Please go to https://github.com/ and sign-up your own account.

A) As the owner of a remote repo, you can push local branch to it.

To check the remote repo name:

```
$ git remote -v
```

```
sashimi@glass:ethtool (pamo-master-test) $ git remote -v
origin  git://git.kernel.org/pub/scm/network/ethtool/ethtool.git (fetch)
origin  git://git.kernel.org/pub/scm/network/ethtool/ethtool.git (push)
pamo    https://github.com/pamolloy/ethtool.git (fetch)
pamo    https://github.com/pamolloy/ethtool.git (push)
```

To add upstream repo for working branch and push it to upstream branch:

```
# List the git branch and their upstream branch information
$ git branch -vv
```

```
sashimi@glass:ethtool (pamo-master-test) $ git branch -vv
  master          d605b92d9a94 [origin/master: ahead 1] ethtool: edit README and add hello world text
  pamo-master     385e8d87591a [pamo/master: ahead 3] README: my 3rd edit
* pamo-master-test 1faeb06eb305 README: Add additional information.
```

```
# Since we are at "pamo-master-test" branch and it does not have
# an upstream branch, it is not allowed to perform git push here.
$ git push
```

```
sashimi@glass:ethtool (pamo-master-test) $ git push
fatal: The current branch pamo-master-test has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin pamo-master-test
```

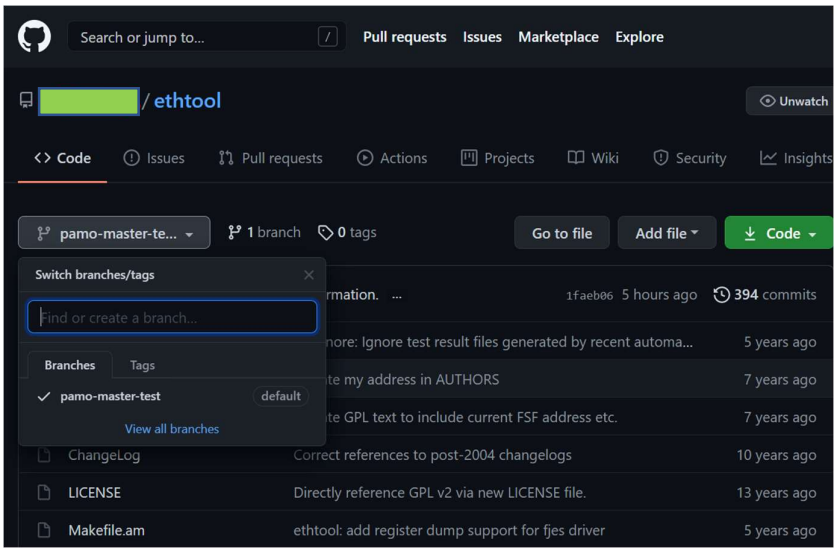
```
# In your GitHub account, create a repo called 'ethtool'. Then,
# add a new tracked repo to it
$ git remote add mygithub https://github.com/<user>/ethtool.git
$ git remote -v
```

```
mygithub        https://github.com/[redacted]/ethtool.git (fetch)
mygithub        https://github.com/[redacted]/ethtool.git (push)
origin  git://git.kernel.org/pub/scm/network/ethtool/ethtool.git (fetch)
origin  git://git.kernel.org/pub/scm/network/ethtool/ethtool.git (push)
pamo    https://github.com/pamolloy/ethtool.git (fetch)
pamo    https://github.com/pamolloy/ethtool.git (push)
```

```
# Then, push "pamo-master-test" branch to "mygithub"
$ git push --set-upstream mygithub pamo-master-test
```

```
Enumerating objects: 1464, done.
Counting objects: 100% (1464/1464), done.
Delta compression using up to 32 threads
Compressing objects: 100% (504/504), done.
Writing objects: 100% (1464/1464), 434.23 KiB | 12.41 MiB/s, done.
Total 1464 (delta 1022), reused 1355 (delta 958)
remote: Resolving deltas: 100% (1022/1022), done.
To https://github.com/[redacted]/ethtool.git
 * [new branch]                pamo-master-test -> pamo-master-test
Branch 'pamo-master-test' set up to track remote branch 'pamo-master-test' from 'mygithub'.
```

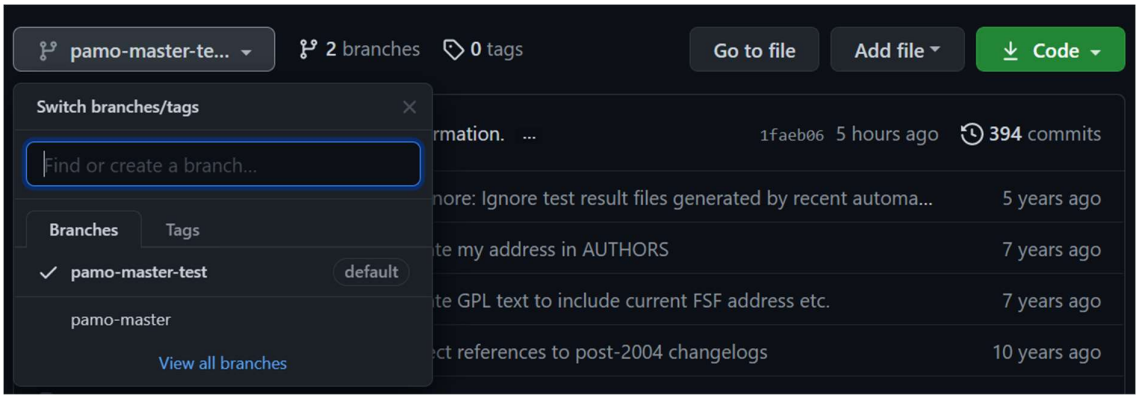
You can cross check what you have just pushed in your GitHub account.



You can also push different upstream branch name instead of local branch name.

```
# To push 'pamo-master-test' as 'pamo-master' to upstream repo
$ git push mygithub pamo-master-test:pamo-master

Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'pamo-master' on GitHub by visiting:
remote:      https://github.com/[redacted]/ethtool/pull/new/pamo-master
remote:
To https://github.com/[redacted]/ethtool.git
 * [new branch]                pamo-master-test -> pamo-master
```

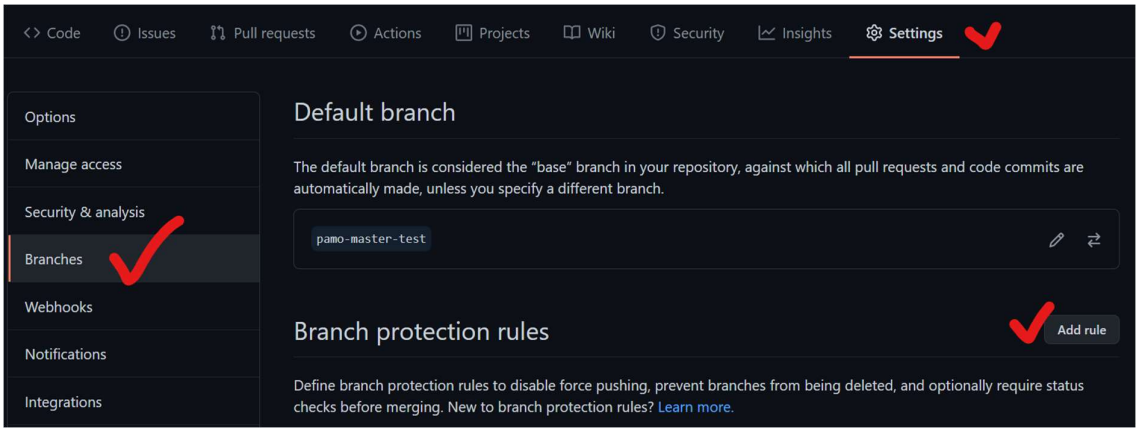
Now, your GitHub account has the new "`pamo-master`" branch



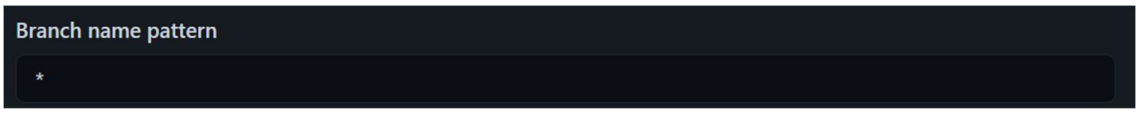
As the owner to the upstream repo, you can delete a branch, say "`pamo-master-test`". However, GitHub default setting prevents you from doing so. At this point of time, you will get below warning when you try to perform '`git push myhithub :pamo-master-test`' to delete the '`pamo-master-test`' branch.

```
sashimi@glass:ethtool (pamo-master-test) $ git push mygithub :pamo-master-test
To https://github.com/elvinongbl/ethtool.git
 ! [remote rejected]           pamo-master-test (refusing to delete the current branch: refs/heads/pamo-master-test)
error: failed to push some refs to 'https://github.com/elvinongbl/ethtool.git'
```

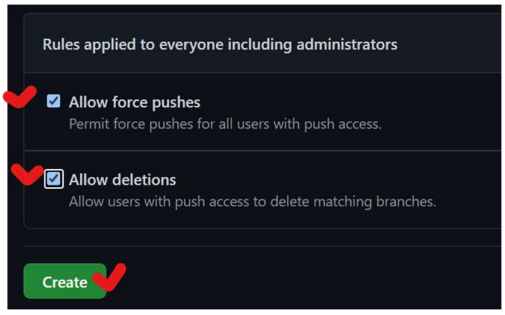
We will need to change the "Branch protection rules" under the repo "Settings" page.



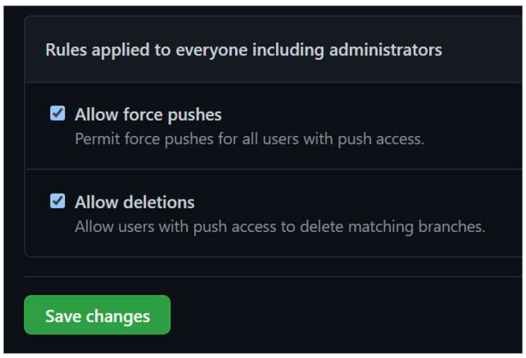
Use * in 'Branch name pattern' to apply the rules to all branches.



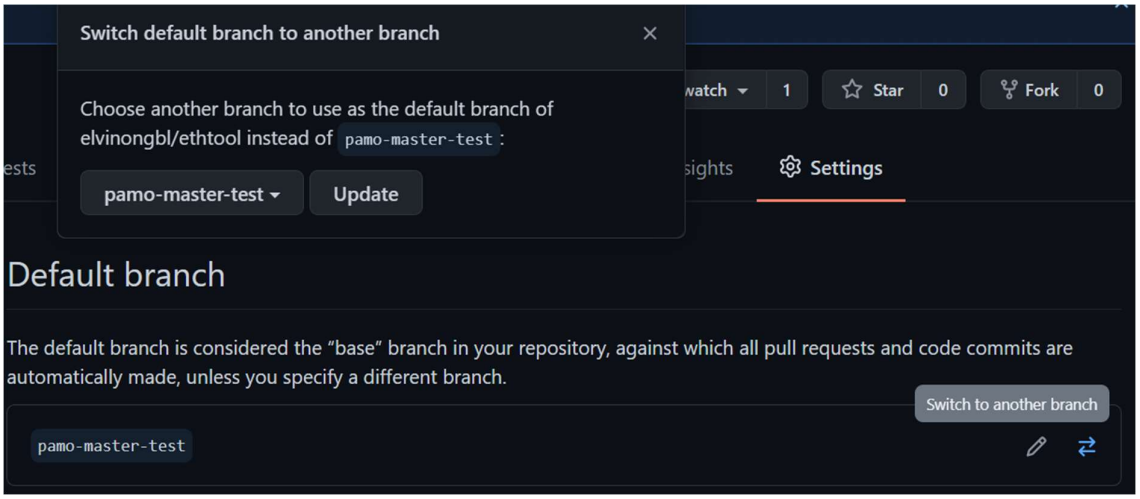
Mark the below options to allow owner to perform force pushes or delete branch



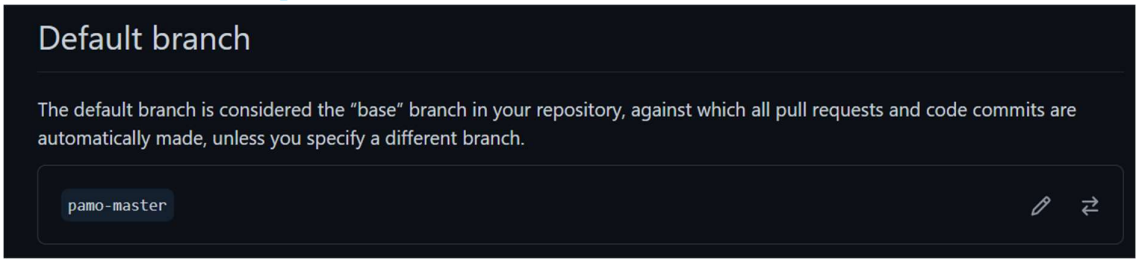
Then, press the 'Save changes' button



Next, you also need to change the default branch that was set to 'pamo-master-test' to 'pamo-master' because you are not allowed to delete a default branch.



Default branch is now 'pamo-master'.



Let's delete the 'pamo-maste-test' branch again.

```
# To delete 'pamo-master-test', simply push empty branch to
# 'pamo-master-test' branch
$ git push --force mygithub :pamo-master-test
sashimi@glass:ethtool (pamo-master-test) $ git push mygithub --force :pamo-master-tes
To https://github.com/[redacted]/ethtool.git
 - [deleted]                   pamo-master-test

# Update mygithub to sync with upstream repo
$ git fetch mygithub
$ git branch -a
sashimi@glass:ethtool (pamo-master-test) $ git branch -a
  master
  pamo-master
* pamo-master-test
  remotes/mygithub/pamo-master
  remotes/origin/HEAD -> origin/master
  remotes/origin/ethtool-3.4.y
  remotes/origin/master
  remotes/origin/next
  remotes/pamo/master
```

In local git repo (after git getch), local 'pamo-master-test' branch that was tracking an already deleted upstream branch 'pamo-master-test' with be marked as 'gone'.

So, we can change the upstream branch for 'pamo-master-test' as follow.

```
# List the git branch upstream branch
$ git branch -vv
sashimi@glass:ethtool (pamo-master-test) $ git branch -vv
  master           d605b92d9a94 [origin/master: ahead 1] ethtool: edit README and add hello world text
  pamo-master      385e8d87591a [pamo/master: ahead 3] README: my 3rd edit
* pamo-master-test 1faeb06eb305 [mygithub/pamo-master-test: gone] README: Add additional information.

# To change the local 'pamo-master-test' branch' to track upstream
# 'mygithub/pamo-master' branch
$ git branch -u mygithub/pamo-master pamo-master-test
$ git branch -vv
sashimi@glass:ethtool (pamo-master-test) $ git branch -vv
  master           d605b92d9a94 [origin/master: ahead 1] ethtool: edit README and add hello world text
  pamo-master      385e8d87591a [pamo/master: ahead 3] README: my 3rd edit
* pamo-master-test 1faeb06eb305 [mygithub/pamo-master] README: Add additional information.
```

Let's say we make change on HEAD commit and then want to git push it to upstream repo:

```
# Change git commit meesage
$ git commit --amend
Author: Ong Boon Leong <[redacted]>
Date:   Fri May 14 11:21:48 2021 +0800

    README: Add additional information.

    We modified the git commit so that we can demonstrate git force push

    Signed-off-by: [redacted]

# push the HEAD commit to upstream repo
$ git push mygithub HEAD:pamo-master
To https://github.com/[redacted]/ethtool.git
 ! [rejected]                HEAD -> pamo-master (non-fast-forward)
error: failed to push some refs to 'https://github.com/[redacted]/ethtool.git'
hint: Updates were rejected because a pushed branch tip is behind its remote
hint: counterpart. Check out this branch and integrate the remote changes
hint: (e.g. 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

# Obviously, the HEAD commit of local branch is in conflict
# with HEAD commit in upstream branch, so to overwrite the
# HEAD commit at upstream branch we will need to force push it over
$ git push --force mygithub HEAD:pamo-master
sashimi@glass:ethtool (pamo-master-test) $ git push --force mygithub HEAD:pamo-master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 32 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 445 bytes | 445.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/[redacted]ethtool.git
 + 1faeb06eb305...fa1dcab2a5f3 HEAD -> pamo-master (forced update)
```

In this section, we have learned about deleting git branch or force push (over-writing) commit history by enabling this special permission in GitHub repo setting. In large project whereby a repo is followed by many developers, tampering with git branch and commit history is often not desirable. So, please understand the implication and use these techniques with care.

B) As "contributor" to an upstream repo, you can send pull-request to share your commit changes.

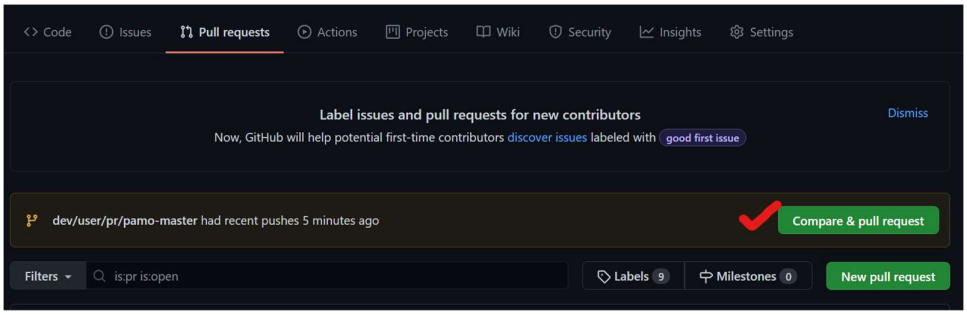
Let's create to new commits as follow:

```
# Make a new commit by changing alexa.txt as follow:
$ vim alexa.txt
Good Morning to Alexa.

Alexa is a smart system that can understands how
ethtool is used and suggest command tips.
$ git add alexa.txt
$ git commit -s -m "alexa: add additional description"

# Make a new commit by changing new_file.txt as follow:
$ vim new_file.txt
Good Night World!
$ git add new_file.txt
$ git commit -s -m "new_file: Change to Good Night World"

# List git history
$ git log --oneline
2b016196295e (HEAD -> pamo-master-test) new_file: Change to Good Night World
f5c7c1b999d0 alexa: add additional description
fa1dcab2a5f3 (mygithub/pamo-master) README: Add additional information.

# Show git status
$ git status
On branch pamo-master-test
Your branch is ahead of 'mygithub/pamo-master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
# Note that git status informs us that local branch is ahead of upstream
# branch by 2 commits. This matches what was shown in git log above too
```

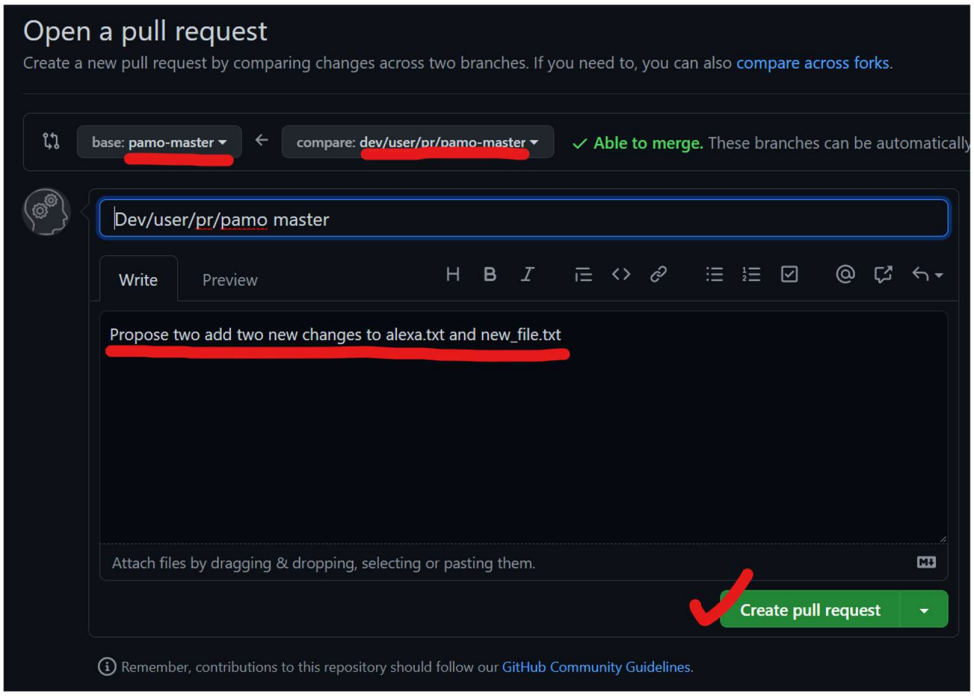
Push a developer's topic branch to GitHub

```
# Push a developer's topic branch to GitHub
$ git push mygithub HEAD:dev/user/pr/pamo-master
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 32 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 713 bytes | 713.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
remote:
remote: Create a pull request for 'dev/user/pr/pamo-master' on GitHub by visiting:
remote:      https://github.com/[redacted]/ethtool/pull/new/dev/user/pr/pamo-master
remote:
To https://github.com/[redacted]/ethtool.git
 * [new branch]                HEAD -> dev/user/pr/pamo-master
```

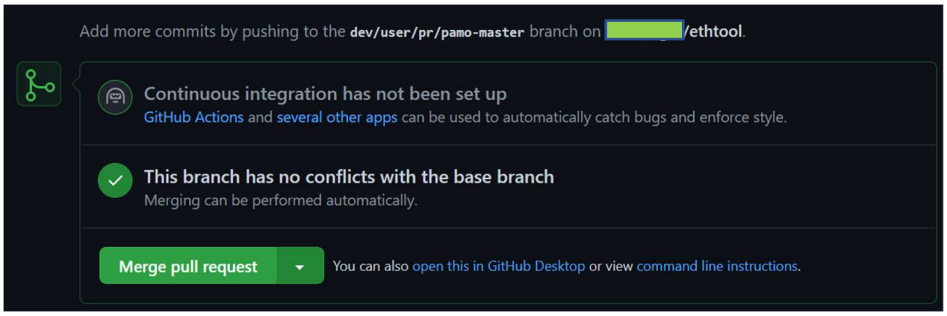
Then, got to GitHub to create a pull-request for the developer's topical branch at `https://github.com/<user>/ethtool/pulls`



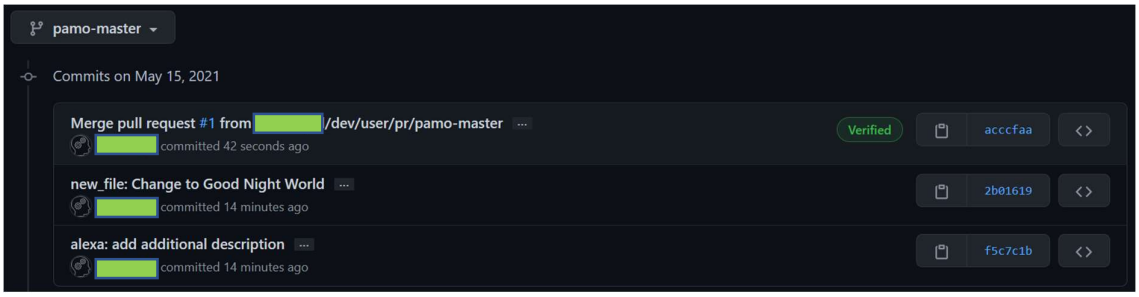
Write-up the pull-request and create pull request.



Here, since you are the owner of the GitHub repo, you can comment on the changes to request the developer to clarify/fix the pull request or if you are satisfied with the PR you can just click "Merge pull request" then "Confirm nerge".



With merge approved, now the git commit history shows the two commits that have been added.



Now, go back to your local git repo and check the git status again.

```
# Show git status
$ git status
On branch pamo-master-test
Your branch is ahead of 'mygithub/pamo-master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

# Now, you see that your local repo is yet to be in sync with
# upstream repo. So, we fetch the latest git states
$ git fetch mygithub
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 626 bytes | 626.00 KiB/s, done.
From https://github.com/[redacted]/ethtool
   fa1dcab2a5f3..acccfaa0d4fa  pamo-master -> mygithub/pamo-master

# Show git status
$ git status
On branch pamo-master-test
Your branch is behind 'mygithub/pamo-master' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean

# Now, git pull from updated upstream branch
$ git pull mygithub pamo-master
From https://github.com/[redacted]/ethtool
 * branch                    pamo-master -> FETCH_HEAD
Updating 2b016196295e..acccfaa0d4fa
Fast-forward

# List git history
$ git log --oneline
acccfaa0d4fa (HEAD -> pamo-master-test, mygithub/pamo-master) Merge pull request #1 from elvinongbl/dev/user/pr/pamo-master
2b016196295e (mygithub/dev/user/pr/pamo-master) new_file: Change to Good Night World
f5c7c1b999d0 alexa: add additional description
fa1dcab2a5f3 README: Add additional information.
87cda0169d70 README: extra changes that causes conflict next
cb3fe007855a alexa: Good morning message
ec46a7ba1865 new_file: initial version of the new_file that says hello world
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
```

```
# Instead of doing all the above, to keep your local branch in-syn
# you could just perform 'git pull --rebase' which include git fetch.
$ git pull --rebase
```

C) As "contributor" to an upstream repo, you can send patch-series to share your commit changes.

In Appendix C, there is an example of bash script that I used for sending patch-series to Linux mailing list. You are encouraged to read through the simple bash script to get an idea what it is doing and after that change it to your own need.

Sending patch-series to mailing list involve below steps:

i. Generate patch series (using '`git format-patch`')
ii. If there are more than 1 patches, write-up cover-letter (in 0000-cover-letter.patch)
iii. Check your cover-letter and patch series are in good shape
iv. Send the patch series using '`git send-email`'. Note also we have setup [sendemail] configuration in ~/.gitconfig earlier.

Identify the commit ID for the patch-series.

```
# List git history
$ git log --oneline
accfaa0d4fa (HEAD -> pamo-master-test, mygithub/pamo-master) Merge pull request #1 from elvinongbl/dev/user/pr/pamo-master
2b016196295e (mygithub/dev/user/pr/pamo-master) new_file: Change to Good Night World
f5c7c1b999d0 alexa: add additional description
fa1dcab2a5f3 README: Add additional information.
87cda0169d70 README: extra changes that causes conflict next
cb3fe007855a alexa: Good morning message
ec46a7ba1865 new_file: initial version of the new_file that says hello world
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
```

Generate patch-series and send-email bash script

```
# Let's say we want to send patches from ec46a7ba1865 to 2b016196295e
# Note the '^' at the back of ec46a7ba1865 to include.

# Next, we use the convenient bash script to generate
# cover-letter, patches and send-email bash script
# ./bformat-patch.sh $START_ID $END_ID $SERIES_VER $TOPIC
$ ./bformat-patch.sh ec46a7ba1865^ 2b016196295e 1 ethtool-text
sashimi@glass:ethtool (pamo-master-test) $ ./bformat-patch.sh ec46a7ba1865^ 2b016196295e 1 ethtool-text
EXT-ethtool-text-v1/0000-cover-letter.patch
EXT-ethtool-text-v1/0001-new_file-initial-version-of-the-new_file-that-says-h.patch
EXT-ethtool-text-v1/0002-alexa-Good-morning-message.patch
EXT-ethtool-text-v1/0003-README-extra-changes-that-causes-conflict-next.patch
EXT-ethtool-text-v1/0004-README-Add-additional-information.patch
EXT-ethtool-text-v1/0005-alexa-add-additional-description.patch
EXT-ethtool-text-v1/0006-new_file-Change-to-Good-Night-World.patch

$ ls
# Note that the bformat-patch.sh script generates a folder "EXT-<topic>-
v1" and sendemail bash script 'EXT-ethtool-text-sendemail.sh'.

sashimi@glass:ethtool (pamo-master-test) $ ls
alexa.txt         COPYING         ethtool.spec.in              ixgb.c            new_file.txt  sfc.c          test-common.c
amd8111e.c        de2104x.c       EXT-ethtool-text-sendemail.sh  ixgbe.c           NEWS          sff-common.c   test-features.
at76c50x-usb.c    e1000.c         EXT-ethtool-text-v1          ixgbevf.c         pcnet32.c     sff-common.h   tg3.c
AUTHORS           e100.c          fec_8xx.c                    LICENSE           qsfp.c        sfpdiag.c      tse.c
autogen.sh        et131x.c        fjes.c                       Makefile.am       qsfp.h        sfpid.c        vioc.c
bformat-patch.sh  ethtool.8.in    ibm_emac.c                   marvell.c         README        smsc911x.c     vmxnet3.c
ChangeLog         ethtool.c       igb.c                        natsemi.c         realtek.c     stmmac.c
configure.ac      ethtool-copy.h  internal.h                   net_tstamp-copy.h  rxclass.c     test-cmdline.c
```

Write the cover-letter (EXT-ethtool-text-v1/0000-cover-letter.patch)

```
# Add description to cover letter
$ vim EXT-ethtool-text-v1/0000-cover-letter.patch

# The original cover letter format.
# Add your email title at *** SUBJECT HERE ***
# Add your patch series description at *** BLURB HERE ***
From 2b016196295ea8c2fda4e40f36e2f01b92292bc2 Mon Sep 17 00:00:00 2001
From: Ong Boon Leong <[redacted]>
Date: Sat, 15 May 2021 17:32:20 +0800
Subject: [PATCH net 0/6] *** SUBJECT HERE ***

*** BLURB HERE ***

Ong Boon Leong (6):
  new_file: initial version of the new_file that says hello world
  alexa: Good morning message
  README: extra changes that causes conflict next
  README: Add additional information.
  alexa: add additional description
  new_file: Change to Good Night World

 README       | 6 ++++++
 alexa.txt    | 4 ++++
 new_file.txt | 1 +
 3 files changed, 11 insertions(+)
 create mode 100644 alexa.txt
 create mode 100644 new_file.txt

# Example of cover letter write-up
From 2b016196295ea8c2fda4e40f36e2f01b92292bc2 Mon Sep 17 00:00:00 2001
From: Ong Boon Leong <[redacted]>
Date: Sat, 15 May 2021 17:32:20 +0800
Subject: [PATCH net 0/6] ethool: Add extra documentations

Hi,

This patch series adds 6 patches to improve the general documentations
of the project. The summary of the changes are:

1/6: Add new_file.txt for XYZ purpose
2/6: Add alexa.txt for ABC reason
3/6-4: Add more description to README
5/6: Fix up some area in alexa.txt
6/6: Change new_file to say Good Night.

These patches have been build-tested and have passed some additional
self-test (if there is relevant).

Please review and provide feedback where seems fit.

Thanks
BL

Ong Boon Leong (6):
  new_file: initial version of the new_file that says hello world
  alexa: Good morning message
  README: extra changes that causes conflict next
  README: Add additional information.
  alexa: add additional description
  new_file: Change to Good Night World

 README       | 6 ++++++
 alexa.txt    | 4 ++++
 new_file.txt | 1 +
 3 files changed, 11 insertions(+)
 create mode 100644 alexa.txt
 create mode 100644 new_file.txt
```

Check your cover-letter and patches are in good-shape. Basically, this means there is no unwanted trailing white-space. The cover letter and commit messages in the patch should normally not exceed 74-characters and they are written with proper indentation. It is also good to see how existing git log on how a good commit looks like.

```
# Inspect cover letter
$ cat 0000-cover-letter.patch

# Inspect patches
$ cat 0001-new_file-initial-version-of-the-new_file-that-says-h.patch
From ec46a7ba1865efe3071b94995fe1597d559d30eb Mon Sep 17 00:00:00 2001
From: Ong Boon Leong <boon.leong.ong@intel.com>
Date: Fri, 14 May 2021 11:41:37 +0800
Subject: [PATCH net 1/6] new_file: initial version of the new_file that says
 hello world

Signed-off-by: Ong Boon Leong <[redacted]>
---
 new_file.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 new_file.txt

diff --git a/new_file.txt b/new_file.txt
new file mode 100644
index 000000000000..557db03de997
--- /dev/null
+++ b/new_file.txt
@@ -0,0 +1 @@
+Hello World
--
2.25.1
```

Note: Observe that a patch is really a git commit. So, sending patch-series in email is really just another way of communicating your commits to the recipient.

Send the cover letter and patch-series to mailing-list.
Note, the sendemail bash script will require some update from you before you use it to send the email and patch-series out.

```
# It is always good to send the patch-series to yourself.
$ vim EXT-ethtool-text-sendemail.sh
#!/bin/bash
# ./EXT-<topic>.sh <folder> to send all patch series
# ./EXT-<topic>.sh <folder>/0001-xyz.patch to send a patch

git send-email $1 \
--to="Some VIP <vip.some@somecopr.com>" \
--to="Another Person <person.another@somecopr2.com>" \
--cc="My Self <self.my@gmail.com>"
# Change it to yourself now and then use the script to send to yourself.

# Note that the comment gives you hint on how to send all patch series or
# just one patch.
# So, you can use the bformat-patch.sh to generate one patch and use the
# the sendemail.sh script to send one patch.
# Finally, to send the cover-letter and patch series
$./EXT-ethtool-text-sendemail.sh EXT-ethtool-text-v1
```

```
EXT-ethtool-text-v1/0000-cover-letter.patch
EXT-ethtool-text-v1/0001-new_file-initial-version-of-the-new_file-that-says-h.patch
EXT-ethtool-text-v1/0002-alexa-Good-morning-message.patch
EXT-ethtool-text-v1/0003-README-extra-changes-that-causes-conflict-next.patch
EXT-ethtool-text-v1/0004-README-Add-additional-information.patch
EXT-ethtool-text-v1/0005-alexa-add-additional-description.patch
EXT-ethtool-text-v1/0006-new_file-Change-to-Good-Night-World.patch

From: Ong Boon Leong <[redacted]>
To: Elvin Ong <[redacted]>
Subject: [PATCH net 0/6] ethool: Add extra documentations
Date: Sat, 15 May 2021 17:58:12 +0800
Message-Id: <20210515095818.9724-1-[redacted]>
X-Mailer: git-send-email 2.25.1
MIME-Version: 1.0
Content-Transfer-Encoding: 8bit

Send this email? ([y]es|[n]o|[e]dit|[q]uit|[a]ll):
```

```
# Always remember to check:
# 1) To add CC list
# 2) Cover letter and the patch series are listed.
# 3) Since this is 1st version, the [PATCH net 0/6] does not show 'v1'.
# 4) Note that in this example 'net' is used in [PATCH net 0/6] and
#    this is for Linux networking subsystem label.
# Note: always remember you can change the bformat-patch.sh to fit your
#       need before using it.
# Finally, to confirm that you want to send the cover-letter and patch
# series, press 'a'.
# If you found issue, and you want to abort, press 'q' to quit.

# Since we send the patches to ourself, we can look them up in our
# email account.
```

Ong Boon Leong	[PATCH net 6/6] new_file: Change to Good Night World - Signed-off-by: Ong Bo
Ong Boon Leong	[PATCH net 5/6] alexa: add additional description - Signed-off-by: Ong Boon L
Ong Boon Leong	[PATCH net 4/6] README: Add additional information. - We modified the git c
Ong Boon Leong	[PATCH net 2/6] alexa: Good morning message - Signed-off-by: Ong Boon Leo
Ong Boon Leong	[PATCH net 3/6] README: extra changes that causes conflict next - Signed-of
Ong Boon Leong	[PATCH net 1/6] new_file: initial version of the new_file that says hello world
Ong Boon Leong	[PATCH net 0/6] ethool: Add extra documentations - Hi, This patch series adds

```
# After you are happy with it, update the sendemail.sh emails and send
# the patches out to the intended recipient.
```

Lastly, assuming you have downloaded the patches into your working repo and you want to apply the patches to try this out.

```
# Let's create a new test branch
$ git branch test-branch
$ git checkout test-branch
$ git log --oneline
acccfaa0d4fa (HEAD -> test-branch, mygithub/pamo-master, pamo-master-test) Merge pull request #
master
2b016196295e (mygithub/dev/user/pr/pamo-master) new_file: Change to Good Night World
f5c7c1b999d0 alexa: add additional description
fa1dcab2a5f3 README: Add additional information.
87cda0169d70 README: extra changes that causes conflict next
cb3fe007855a alexa: Good morning message
ec46a7ba1865 new_file: initial version of the new_file that says hello world
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name

# Reset the history to "0975dca47038 (pamo/master) Merge pull
# request #1 from pamolloy/features-long-name" so that we can apply
# the 6 patches we just created earlier
```

```
$ git reset --hard 0975dca47038
sashimi@glass:ethtool (test-branch) $ git reset --hard 0975dca47038
HEAD is now at 0975dca47038 Merge pull request #1 from pamolloy/features-long-nam

# List the patch series
$ ls -1 EXT-ethtool-text-v1/
0000-cover-letter.patch
0001-new_file-initial-version-of-the-new_file-that-says-h.patch
0002-alexa-Good-morning-message.patch
0003-README-extra-changes-that-causes-conflict-next.patch
0004-README-Add-additional-information.patch
0005-alexa-add-additional-description.patch
0006-new_file-Change-to-Good-Night-World.patch

# Note that 0000-cover-letter.patch is actually not a patch.
# Finally to apply 0001 patch
$ git am EXT-ethtool-text-v1/0001-new_file-initial-version-of-the-
new_file-that-says-h.patch
sashimi@glass:ethtool (test-branch) $ git am EXT-ethtool-text-v1/0001-new_file-initial-version-of-the-new_file-that-says-h.patc
Applying: new_file: initial version of the new_file that says hello world

# list git history
$ git log --oneline
2bfc5f25e6e9 (HEAD -> test-branch) new_file: initial version of the new_file that says hello worl
0975dca47038 (pamo/master) Merge pull request #1 from pamolloy/features-long-name
30f1d53fa4df Also support passing the long name as a feature
b888f358763a ethtool: Support for configurable RSS hash function

# You can apply more patches to try out 'git am' here.
# For now, let's just remove the 'test-branch'. In order to do that
# you need to switch to another branch
$ git branch
  master
  pamo-master
  pamo-master-test
* test-branch

$ git checkout pamo-master-test
$ git branch -D test-branch
Deleted branch test-branch (was 2bfc5f25e6e9).
$ git branch
  master
  pamo-master
* pamo-master-test
```

Finally, congratulations! You are now ready to be an expert user of git tool.

If you like this git workshop material, please drop me a thank you note!

Appendix A: Network Proxy Setting at ~/.bashrc

```
# Add below global environment exports to local bash session
$ vi ~/.bashrc

# You need to check with your network administration for
# network proxy information
```

```
export SOCKS_SERVER=<proxy-server.com>:<SOCKS Port>
export HTTP_PROXY=<proxy-server.com>:<HTTP Port>
export HTTPS_PROXY=<proxy-server.com>:<HTTPS Port>
export FTP_PROXY=<proxy-server.com>:<FTP Port>
export HTTP_DIRECT=localhost,127.0.0.0/8,172.169.0.0/20,192.168.0.0/16
export SOCKS_DIRECT=$HTTP_DIRECT
export NO_PROXY=$HTTP_DIRECT
export ALL_PROXY=$HTTP_DIRECT
export socks_server=$SOCKS_SERVER
export http_proxy=$HTTP_PROXY
export https_proxy=$HTTPS_PROXY
export ftp_proxy=$FTP_PROXY
export http_direct=$HTTP_DIRECT
export socks_direct=$SOCKS_DIRECT
export no_proxy=$NO_PROXY
export all_proxy=$ALL_PROXY

PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u\[\033[00m\]@\[\033[01;32
m\]\h\[\033[00m\]:\[\033[01;32m\]\W\[\033[00m\]$(__git_ps1 " (%s)") \$ '
```

```
Note: PS1 is the global environment variable that defines shell prompt format.
Note: $(__git_ps1 " (%s)") is a useful shell prompt augmentation to print the git
branch if your current working directory is a git repository.
```

Appendix B: Git configuration at ~/.gitconfig

```
$ vim ~/.gitconfig
[user]
email = felix.ong@gmail.com
name = Felix Ong

[sendemail]
from = "Felix <felix.ong@gmail.com>"
smtpServer = smtp.gmail.com
smtpUser = felix.ong@gmail.com
smtpEncryption = tls
smtpServerPort = 587
smtpPass = <gmail token>
signedoffcc = false
suppresscc = all
chainreplyto = false
assume8bitEncoding = utf-8
confirm = always

[tig "color"]
# A strange looking cursor line
cursor = yellow default bold

# Title colors <foreground background attribute>
title-blur = black white
title-focus = red white bold

# Diff colors <foreground background attribute>
diff-header = yellow default
diff-index =  green default
diff-chunk =  red default

date = white black
author = green black
directory = white black
file = green black

# View-specific color
[tig "color.tree"]
date = green default bold

[color]
diff = auto
ui = auto
interactive = auto
grep = always

[color.branch]
current = green
local = white
remote = red
upstream = yellow
plain = white

[color "grep"]
    match = red

[core]
abbrev = 12
editor = vim
# Uncomment for network proxy
#gitproxy = /usr/bin/git_proxy_command
whitespace = trailing-space,tab-in-indent
pager = sed 's/\t/>-------/g' | less -R
```

```
[alias]
 co = checkout
 dc = describe --contains
 br = branch -v
 ci = commit
 st = status
 sts = status -sb
 ol = log --oneline
 olf = log --pretty=format:\"%h [%ci %cn]: %s\"
 cp = cherry-pick
 showf = show --format=fuller
 logf = log --format=fuller
 merge-log = "!f() { git log --stat \"$1^..$1\"; }; f"
 merge-ol = "!f() { git log --oneline \"$1^..$1\"; }; f"
 merge-olg = "!f() { git log --oneline --graph \"$1^..$1\"; }; f"
 bl = blame --abbrev=11
```

Note: <gmail token> is obtained from **Google Account -> Security -> Signing in to Google : App passwords.**

Note: "editor = vim" set the default git commit editor to 'vim'. I encourage you to learn it. If you are not familiar with 'vim', you could change it to 'editor = nano' or 'editor = gedit'.

Note: [alias] contains a list of git command alias (short-cut) that you can use in bash shell prompt. For example, instead of typing 'git log --oneline' which often you may mistype as 'online' (because our brain trick us there all the times), we could just type 'git ol'.

```
# Add below global environment exports to local bash session
$ vim ~/common/bin/socatproxy
```

```
#!/bin/sh
# Please choose your site proxy server accordingly
exec socat stdio SOCKS:<proxy-server@somename.com>:$1:$2
```

Note: proxy-server@somename.com is the secure socket network server

Appendix C: bash script for patch-series

```
$ vim bformat-patch.sh
#!/bin/bash

# ./bformat-patch.sh $START_ID $END_ID $SERIES_VER $TOPIC
#
# For version 1 of patch-series
# ./bformat-patch.sh fec4d42724a1 ae4393dfd472 1 vlan-filter

# For version 2 of patch-series
# ./bformat-patch.sh 5afe69c2ccd0 38318f23a7ef 2 vlan-filter

START_ID=$1
END_ID=$2
SERIES_VER=$3
TOPIC=$4

# Usually, the send email list is fixed, so
# we just need to generate once for version 1
gen_sendemail() {
cat << EOF > $1
#!/bin/bash
# ./EXT-<topic>.sh <folder> to send all patch series
# ./EXT-<topic>.sh <folder>/0001-xyz.patch to send a patch

git send-email \$1 \\
--to="Some VIP <vip.some@somecopr.com>" \\
--to="Another Person <person.another@somecopr2.com>" \\
--cc="My Self <self.my@gmail.com>"
EOF
}

SENDMAIL=EXT-${TOPIC}-sendemail.sh
PATCHDIR=EXT-${TOPIC}-v${SERIES_VER}

# 'net' is Mailing List(ML) tag for Linux networking subsystem
ML=net

# In Linux M/L, we can send Request For Comment(RFC) or Patch
PATCHTYPE=PATCH
#PATCHTYPE=RFC

# We can reuse the send-email list
if [ x"$SERIES_VER" == x"1" ]; then
    git format-patch ${START_ID}..${END_ID} --cover-letter \
        --subject-prefix="$PATCHTYPE $ML" -o $PATCHDIR
    gen_sendemail $SENDMAIL
    chmod a+x $SENDMAIL
else
    git format-patch ${START_ID}..${END_ID} --cover-letter \
        --subject-prefix="$PATCHTYPE $ML v${SERIES_VER}" -o $PATCHDIR
fi

FOR_LINUX=false
if [ x"$FOR_LINUX" == x"true" ]; then

    # Linux project has get_maintainer.pl script to list the mailing list
    # recipients for a particular patch. So, it is good to cross-check
    # the list in gen_sendemail() is accurate
    echo "Show maintainer list and PLEASE cross check if $SENDMAIL is READY"
    echo "================================================================="

    SERIES=$(ls -1 $PATCHDIR -I0000-cover-letter.patch)
    for patch in $SERIES; do
       echo -e "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
       ./scripts/get_maintainer.pl $PATCHDIR/$patch | sed -e "s# (#\t\t\t(#g"
       echo -e "-----------------------------------------------------------"
    done
    echo "================================================================="
    echo "After you have sent, you check at patchwork at \
https://patchwork.kernel.org/project/netdevbpf/list/"
    echo "================================================================="
fi
```

Appendix D: vim editor configuration ~/.vimrc

```
$ vim ~/.vimrc
colorscheme desert
set background=dark
set nonumber
set listchars=tab:>-,nbsp:#
set list
set hlsearch
set tabstop=8
highlight ExtraWhitespace ctermbg=red guibg=red
match ExtraWhitespace /\s\+$/
set tw=74
set fo+=t

" No annoying sound on errors
set noerrorbells
set novisualbell
```

To mark TAB as >------

```
listchars=tab:>-,nbsp:#
```

To highlight trailing and leading white-space

```
highlight ExtraWhitespace ctermbg=red guibg=red
match ExtraWhitespace /\s\+$/
```

To limit text width to 74 character.

```
set tw=74
```