Steps for contributing to Seismic Julia

(1) Create a **GitHub account**.

(2) Got to https://github.com/SeismicJulia/Seismic.jl and **fork the repository** into your account by clicking the “Fork” button.

(3) In your local machine, create a **directory** where you would like to have the **Seismic Julia source codes**. For example:
```
~$ mkdir SeismicJulia.
```

(4) **Config your username and email** in your local machine. It is a good idea to use the same email as in the gihub account:
```
git config --global user.name "your name"
git config --global user.email "your_email@example.com"
```

(5) Browse to the new directory and **clone the repository** you have forked in (1) into your local machine:
```
~$ cd SeismicJulia
~$ git clone https://github.com/your_github_account/Seismic.jl.git
```

(6) Check the repository content in your machine:
```
~$ cd Seismic.jl
```
`~$ ls -la` (this will show that the package is now in your machine)

(7) Check the **link** between your machine and your repository uin GitHub.
```
~$ git remote -v
```
This will show:
```
origin     https://github.com/your_github_account/Seismic.jl.git (fetch)
origin     https://github.com/your_github_account/Seismic.jl.git (push)
```
Here, `origin` is the name of your remote repository in GitHub.

(8) To keep the repository updated with the one forked (the base fork) and include the changes made by others in the Seismic Julia package, we need to **sync the fork** of the repository to keep it up-to-date with the upstream base repository. First **configure a remote** that points to the **upstream repository** with git:
```
git remote add upstream https://github.com/SeismicJulia/Sesmic.jl.git
```

(9) Verify the new **upstream repository** you've specified for your fork:
```
~$ git remote -v
origin     https://github.com/your_github_account/Seismic.jl.git (fetch)
origin     https://github.com/your_github_account/Seismic.jl.git (push)
upstream   https://github.com/SeismicJulia/Seismic.jl.git (fetch)
upstream   https://github.com/SeismicJulia/Seismic.jl.git (push)
```

(10) **Fetch** the branches and their respective commits from the **upstream repository**. Commits to `master` will be stored in a local branch, `upstream/master`:
```
~$ git fetch upstream
```

(11) Now you have two choices.

  (a) You can **update your fork** from the upstream repository. Note that this will destroy the changes in your local repository in favor of the content in the SeismicJulia package:
```
~$ git pull upstream master
```
  (b) Or you can **merge** the changes from `upstream/master` into your local master branch. This brings your fork's master branch into sync with the upstream repository, without losing your local changes.:
```
~$ git merge upstream/master
```
  If you If you have a **conflict** in your files afeter merging, you will need to resolve them. Resolving conflicts is quite simple, you just edit the files in conflict, remove the conflict makers, and keep the version of the file you prefer. You can eventually need to add or delete e file to solve a conflict. Where is explained how to solve these conflicts: https://help.github.com/articles/resolving-a-merge-conflict-from-the-command-line/

Note: The `git pull` command is a combination of `fetch` and `merge`.

(12) In both cases in (11), you need now to **push these changes** to your own remote repository:
```
~$ git push -u origin master
```
This will "upload" the changes to the remote repository. Here `-u` is used to remember the parameters `origin` (remote repository name) and `master` (local branch name).

(13) We want to develop and constantly change the package. So if **Seismic.jl** is installed in julia in your local machine, we first need to **remove** it:
```
~$ julia
julia> Pkg.rm("Seismic")
```

(14) Then, we need to **export the path to Seismic Julia codes** in the bash profile. Add these lines to your bash configuration file:
```
# Path to Seismic.jl
export SEISMIC_PATH=path_to_Seismic.jl
export JULIA_LOAD_PATH=$SEISMIC_PATH/src
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$SEISMIC_PATH/src/Imaging/c
```

We can finally start to make changes and commit them

(15) Now our local (and remote) packages are finally up-to-date with SeismiJulia and we are using the package we are developing. We can finally start to make changes to the Seismic.jl and commit them to the base fork (upstream). It is useful here to take a look at git tutorials to decide if we need to work on a **new branch** instead of `master`. For example:
```
~$ git branch my_branch_name
```
By this means, we created a new branch where we are going to work in an especific feature of the code. We can create several branches like this. Then, when the feature we are working on in a particular branch is done, we can merge it with the master and push it to the remote server.

(16) If you want to work in a new branch, you need to change to that branch using git **checkout**:
```
~$ git checkout my_branch_name
```

(17) Perhaps is better to do (15) and (16) all at once. This will **create and checkout a new_branch** all at once:
```
~$ git checkout -b my_branch _name
```

(18) To **merge** `my_branch_name` with the `master` branch. Be careful that before this you should first checkout into `master` branch with `git checkout master`. Then do:
```
~$ git merge my_banch_name
```
Again, a **conflict** can arise if you happen to change the same line of the same file using different branches. Look at (11b) how to resolve the conflicts, it is not difficult.

(19) **Delete the branch** `my_branch_name` since we finish using it:
```
~$ git branch -d my_branch name
```

(20) **Delete a bad branch** you don't want to merge:
```
~$ git branch -d --force bad_branch_name
```
or equivalently:
```
~$ git branch -D bad_branch_name
```

(21) **Track** all your files where the dot stands so everything in and beneath it is added. This is neccesary if you created new files. They will be untracked and you need to track them before commiting.:
```
~$ git add -A
```
or equivalently:
```
~$ git add --all
```

(22) **Commit all your changes** and prepare to push them:
```
~$ git commit -a
```
With this command option `-a` the git commit command will autoremove deleted files in the commit. A nice usage would be:
```
~$ git commit -am "Delete stuff"
```

(23) **Add a message** for your commit, for example:
```
~$ git commit -m "This is my first push to update the package"
```

(24) Or do (20) and (21) **all at once**:
```
~$ git commit -am "This is my first push to update the package"
```

(25) **Push** your changes to your remote (forked) repository:
```
~$ git push
```

(26) Finally, **make a pull request** through GitHub. For this, browse to `https://github.com/`*`your_github_account`*`/Seismic.jl` and check that your recent commit is already there. Then:
(a) Click “New pull request”
(b) Click “Creat pull request”
(c) Add a title for the pull request and a message explaining what you are adding or changing to the package.
(d) Review the Pull request.
(e) Send the request.
(f) The owner/s of the package will review the pull rquest and merge it with the main code.

Done!