Please, before sending patches, read these brief comments. They are here to

help the project have both its users happy using the program and the

developers/maintainers feel good when trying to change code that other

people contributed.

For the record, when this document mentions "I", it mostly means Rogério

Brito's (@rbrito) is the one to blame.

# Write good commit messages

When you write your pull request and your commit messages, please, be

detailed, explaining why you are doing what you are doing. Don't be afraid

of being too verbose here. Also, please follow the highly recommended

guidelines on how to write good [good commit messages][commit-msgs].

When in doubt, follow the model of the Linux kernel commit logs. Their

commit messages are some of the best that I have seen. Also, the ffmpeg has

some good messages that I believe that should be followed. If you are in a

hurry, read the section named

["Contributing" from subsurface's README][contributing].

[commit-msgs]: https://robots.thoughtbot.com/5-useful-tips-for-a-better-commit-message

[contributing]: https://github.com/torvalds/subsurface/blob/master/README#L71-L114

# Test that your changes don't break existing functionality

Make sure that you have all dependencies installed, like via:

pip install -r requirements.txt

pip install -r requirements-dev.txt

Run the test suite with

py.test coursera/test -v --cov coursera --cov-report html

If some test fails, please don't send your changes yet. Fix what broke

before sending your pull request.

If you need to change the test suite, explain in the commit message why it

needs to be changed (e.g., the page layout or the authentication methods

from coursera changed, or they implemented a new kind of course).

## Using tox and pyenv

Running only py.test is okay for one-time check, but as it runs on your

currently active Python installation, you're only checking the script on one

Python version. A combination of tox + pyenv will allow you to run tests on many

Python versions easily.

First, install tox (listed in requirements-dev.txt) and pyenv (see their manual:

[pyenv installation](https://github.com/yyuu/pyenv#installation). Make sure to

activate pyenv or start a new shell. Then, install Python versions and run tox:

pyenv install 3.5.1

pyenv install 3.4.4

pyenv install 3.3.6

pyenv install 3.2

pyenv install 2.7.5

pyenv install 2.6.9

pyenv local 3.5.1 3.4.4 3.3.6 3.2 2.7.5 2.6.9

tox

# Check for potential bugs

Please, help keep the code tidy by checking for any potential bugs with the

help of [`flake8`][flake8], and [`pylint`][pylint]. If you know of any

other good tools for analyzing the code, let me know about them!

[flake8]: https://pypi.python.org/pypi/flake8

[pylint]: https://pypi.python.org/pypi/pylint

If you happen to find any issue reported by these programs, I welcome you to

fix them. Many of the issues are usually very easy to fix and they are a

great way to start contributing to this (and other projects in general).

Furthermore, we all benefit from a better code base.

# Changes in the tools that we use

If you are proposing the use of a substitute of a tool that we already use,

take a few paragraphs to tell us why we would like to change.

If we are not using something, it is most likely that one of the following

options applies:

1. I (@rbrito) may not even know that what you are proposing exists or have

not yet "seen the light" as to why I should use it instead of using what

I am currently using.

2. Even if I know about the tool, I may not know how to use it, or how it

would make me more productive. Educate me and we all will gain from a

better project.

----

# Short (and incomplete) cheat-sheet for contributions

## To start contributing

Fork the repository using github (there's a button) and clone \*your\* copy to

your local machine for your personal modifications. Let's say that your

github username is `username`:

```

git clone https://github.com/username/coursera-dl

```

## Making your changes

Initiate a \*new\* branch on your local repository copy.

```

cd coursera-dl

git checkout -b my-new-branch

```

Then, start hacking, doing whatever you want. Really, just \*play\* with what

you want. After you make each "logical chunk" of changes (say, you fixed one

\*specific\* bug, or you made one \*specific\* feature), commit your changes.

### See what you changed

Use the following commands to see which files you changed and which changes

you made, just to recap, before sending them to the public (up to this

moment, everything that you made lives only on your computer):

```

git status

git diff

```

### Select what you want to commit

You may have made more changes than you want to contribute at this point in

time (this is \*very\* common). So, we select which changes we want to commit

with:

```

git add -p

```

Then a little prompt will appear and you will interactively select the

changes that you want to make part of the commit. The main options there are

`y` or `n` for including/not including respectively and `s` to split a very

large change into smaller changes ones (this is a killer feature of `git add

-p`).

### See what changes will make into your commit

To see the changes that you have selected to be part of your commit, use:

```

git diff --staged

```

See if everything is OK. If not, then use `git add -p` to add something that

you may have missed or, if you included something else that you \*didn't\*

intend to include, "un-include" it with:

```

git reset -- file-with-changes-to-un-include

```

Then, you can start over and include only what you meant to put in a single

commit.

### Once you are satisfied with your changes

If you think that your changes make one "logically" cohesive change to the

project, create a commit of them in the repository with:

```

git commit

```

Then, enter a descriptive message of \*what\* (not why, not how) you changed

on the \*first line\* and after that, you are free to write your prose

detailing how you did what you did (e.g., which documentation you read), why

you did what you did (Is that a preparation for a future change? Does it

improve the speed or memory requirements of the project?) and other

comments.

Once again, it is good to have in mind some

[rules for good commit messages][commit-msgs].

### Send your changes to github

To see the commit that you just made (with author, date, short and long

description only), use:

```

git show

```

If you want to see the entire commit, use:

```

git show -p

```

If you are satisfied with your changes, go ahead and type:

```

git push origin my-new-branch

```

Where `my-new-branch` is the branch that you created for this set of changes

(you remember that, right?). If everything goes correctly, then, after a few

seconds, your changes will be right there on the cloud, on github! Yay!

To inform the maintainer of the project that you have some shiny changes

that you are proposing for inclusion on the main project, github will show

you somewhere on \*your\* fork of the project that you have a branch and that

you can create a pull request.

Click that button, compose a message for the maintainer of the project and

tell him/her about what you just did, why you are proposing etc. Send that

message and the maintainer will be notified that you made changes and that

you'd like them to be included in the main project.

That's essentially it.

There are many other tricks/steps in the git workflow, but these are the

basics that I (@rbrito) think that will suffice for a start. If you want a

few details more, feel free to ask me to include them here.

# Release procedure

This section is for project maintainers.

DRAFT

1. Run tests locally. Use tox to run tests on all supported Python versions.

2. Run the script on several courses to check sanity

3. Update CHANGELOG.md, increment version, put what you've added to the

changelog into commit message. This way it gets it way into releases page

with a nice description of the changes.

`git add ... & git ci -m 'Bump version (old\_version -> new\_version)'`

4. `git tag new\_version`

5. `git push && git push --tags`

6. `python setup.py sdist bdist\_wheel --universal` to build the package

7. `twine upload dist/coursera-dl-0.6.1.tar.gz` to deploy the package.

## Docker

Build new Docker image from PyPI package:

```

docker build --tag courseradl/courseradl --build-arg VERSION=0.11.2 .

```

Run the image:

```

docker run --rm -it -v "$(pwd):/courses" -v "$HOME/.netrc:/netrc" courseradl -n /netrc -- google-machine-learning

```

Publish the image:

```

docker push courseradl/courseradl

```