Git/Doxygen/Docker

Ricardo Ribeiro (rfribeiro@ua.pt)

September 2019

1 Git

A version control system, or VCS, tracks the history of changes as people and teams collaborate on projects together. As the project evolves, teams can run tests, fix bugs, and contribute new code with the confidence that any version can be recovered at any time. Git is the most-used VCS in the world (https://guides.github.com/introduction/git-handbook/).

Check if git is installed or what version is installed:

```
$ git --version
```

If you don't have the git package installed, you can install it with:

```
$ sudo apt update
$ sudo apt install git
```

on Linux operating systems. In another operating systems visit https://www.atlassian.com/git/tutorials/install-git.

Now that you have Git installed, you should configure it with your "user.name" and your "user.email" by using the following commands:

```
$ git config --global user.name "Your Name"
$ git config --global user.email "youremail@domain.com"
```

Git embeds this information into each commit you do. You can see all of the configuration items that have been set by using:

```
$ git config --list
```

The information you enter is stored in your Git configuration file, located in `~/.gitconfig`, which you can optionally edit by hand with a text editor.

Now that you have the git installed and configured, you can create a repository using several platforms, such as github, code.ua, among others. However, you can initialize an empty Git repository locally, using:

```
$ git init my-repository
```

To copy an existing Git repository hosted remotely, you can use the following command:

```
$ git clone https://www.github.com/username/repo-name
```

When you have modified or created a file and have marked it to go in your next commit, it is considered to be a staged file. Check the status of your Git repository, including files added that are not staged, and files that are staged, using:

```
$ git status
```

You can stage a specific modified file, which you can run multiple times before a commit, with the following command:

```
$ git add my_script.py
```

You can add all files in the current directory using:

```
$ git add .
```

You can remove a file from staging while retaining changes within your working directory, before a commit, with:

```
$ git reset my_script.py
```

Once you have staged your updates, you are ready to commit them, which will record changes you have made to the repository. You can commit staged files with a message so that you can track commits by using:

```
$ git commit –m "Commit message"
```

If you need to modify your commit message, you can do so with the –amend flag:

```
$ git commit ––amend –m "New commit message"
```

In order to updates the remote repository with any commits made locally, you can use:

```
$ git push
```

You can updates the local lines of development with updates from its remote counterpart:

```
$ git pull
```

A branch in Git is a movable pointer to one of the commits in the repository, it allows you to isolate work and manage feature development and integration.

You can list all current branches (an asterisk (*) will appear next to your currently active branch) with:

```
$ git branch
```

To create a new branch:

```
$ git branch new–branch
```

You will remain on your currently active branch until you switch to the new one. In order to switch to any existing branch and check it out into your current working directory, you can use the following command:

```
$ git checkout new–branch
```

To update the changes to the remote repository, after add and commit, in the new-branch you can use:

```
$ git push ––set–upstream origin new–branch
```

You can merge the specified branch's history into the one you're currently working in using:

```
$ git merge branch–name
$ git push
```

In case there are conflits, you can abort the merge before push using:

```
$ git merge ––abort
```

1.1 Doxygen

Doxygen is the de facto standard tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as C, Objective-C, C#, PHP, Java, Python, etc.

In order to install the doxygen from source you can clone the git repository of the doxygen using:

```
$ git clone https://github.com/doxygen/doxygen.git
```

Before you initiate the installation process, you should have to check if the necessary build tools are installed. In Linux you can proceed as follow:

```
$ sudo apt update
$ sudo apt install build–essential cmake python flex bison libc6
```

After this, enter in the doxygen folder and you should do the following commands:

```
$ mkdir build
$ cd build
$ cmake -G "Unix Makefiles" ..
$ make
```

After the binaries have been built, you can use the next command to install them:

```
$ make install
```

Now, doxygen is installed. For other operating systems visit `http://www.doxygen.nl/download.html`.

To generate a manual for your project you typically need to follow some steps.

You have to write in your code a special comment block, that is a style comment block with some additional markings, so doxygen knows it is a piece of structured text that needs to end up in the generated documentation (`http://www.doxygen.nl/manual/docblocks.html#specialblock`).

Doxygen uses a configuration file to determine all of its settings. Each project should get its own configuration file. A project can consist of a single source file, but can also be an entire source tree that is recursively scanned.

Doxygen can create a template configuration file for you by using:

```
$ doxygen -g <config-file>
```

where config-file is the name of the configuration file. If you omit the file name, a file named Doxyfile will be created. You can open the Doxyfile and edit the values of each tag used. However, you can probably leave the values of most tags in a generated template configuration file to their default value. For more details about the configuration file use the link `http://www.doxygen.nl/manual/config.html`.

In order to generate the documentation o can run the following command:

```
$ doxygen yourDoxyfile
```

1.2 Docker

Docker is a platform for developers and sysadmins to develop, deploy, and run applications with containers. The use of Linux containers to deploy applications is called containerization (`https://docs.docker.com/get-started/`).

In order to install docker, you need to install some prerequisite packages, following in Linux:

```
$ sudo apt-get update
$ sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg-agent \
  software-properties-common
```

Add the Docker GPG key using:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

The next step is adding the Docker APT repository to set up the stable repository:

```
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) \
    stable"
```

You may be prompted to confirm that you wish to add the repository and have the repository's GPG key automatically added to your host. The lsb_release command should populate the Ubuntu distribution version of your host.

Update APT package index again and install the latest version of Docker Engine and containerd:

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

For other operating systems visit `https://docs.docker.com/install/`.
You can test if your installation works by running the simple DOcker Image, hello-world:

```
$ sudo docker run hello-world
```

To list the images that was downloaded to your machine, you can use the following command:

```
$ sudo docker image ls
```

You can also list the containers (spawned by the images):

```
## List Docker containers (running, all, all in quiet mode)
$ sudo docker container ls
$ sudo docker container ls --all
$ sudo docker container ls -aq
```

Now, you can create a Docker Hub account in order to easily share your containers with others (`https://hub.docker.com/`).
Typically, accessing the docker daemon requires root. You can simple do everything as the root user:

```
$ sudo su - root
```

You can check the version installed of the docker in you machine by using:

```
$ docker version
```

if this command run without any errors/warnings, you have access to the docker daemon.
In order to, push images to your Docker Hub account, you have to login:

```
$ docker login
```

A Docker image is a container template from which one or more containers can be run. It is a rooted filesystem that, by definition, contains all of the file dependencies needed for whatever application(s) will be run within the containers launched from it. The image also contains metadata describing options available to the operator running containers from the image.
Most modern programming languages offer an official image. For example, there is an official image for the Python programming language: `https://hub.docker.com/_/python/`
To pull an image off Docker Hub(in this case the official image for Python), you can use the following command:

```
$ docker pull python
```

We use the "docker run" command to run containers from an image. To run a container and attach to it in one command, use the "-it" flags. Here we run bash in a container from the ubuntu image:

```
$ docker run -it ubuntu bash
$ root@4b83b0c5d1fd:/# ls
$ root@4b83b0c5d1fd:/# exit
```

You can also run container we previously created, without an interactive shell:

```
$ docker start container_name
```

And stop the container writing docker stop container_name:

```
$ docker stop container_name
```

If you want to see all running containers, just run:

```
$ docker ps
```

And for all containers we add "-a"at the end of this same command, like this "docker ps -a".
When you run containers, you would also like to know how much resources they are using, for that purpose you can use the command:

```
$ docker stats
```

You can remove a docker container (optionally passing -f to "force" the removal if the container is running) using the container name or id, as follows:

```
$ docker rm -f 4b83b0c5d1fd
```

You can build images creating a Dockerfile. Dockerfile defines what goes on in the environment inside your container.

Assume that you have a folder on your local machine with your code named `classifier` that has the following content:

```
README.md
classify_image.py
entry.sh
```

Go to that directory and create a file called Dockerfile. The following content is an example of a Dockerfile for a classification program written in python that is the referred folder:

```
FROM tensorflow/tensorflow:1.5.0-py3

  # pre-reqs
  RUN apt-get update && apt-get install -y wget

  # our app
  ADD classify_image.py /classify_image.py
  ADD entry.sh /entry.sh
  RUN chmod +x /entry.sh

  # ENTRYPOINT ["python", "/classify_image.py"]
  ENTRYPOINT ["/entry.sh"]
```

Now, you can build your Dockerfile by running the next command:

```
$ docker build --tag=ImageName:TagName .
```

To run the app from the built image, set an environment variable, `$URL`, to a publicly available URL to an image to classify.

For example,

```
  docker run -it --rm -e URL=https://raw.githubusercontent.com/TACC/taccster18_Cloud_T
ImageName:TagName
```

In order to share our container, you need to create a repository in your docker hub account and push the container to the repository. To do this you can run the following commands:

```
$ docker tag local-image:tagname reponame:tagname
$ docker push reponame:tagname
```