Math 4610 Lecture Notes

Using Git to Work Locally *

Joe Koebbe

September 8, 2019

*These notes are part of an Open Resource Educational project sponsored by Utah State University

Math 4610 Contents: Using Git to Work Locally.

In this part of the notes, a brief primer for **git** that will help you get started using repositories locally. You will also learn to clone and pull repositories from Github to work on existing repositories.

In order to efficiently use **git** you will need to be working in a command terminal. You can use Cygwin or the command windows in Windows or on your Apple computer/laptop. This has already been covered in previous lectures. So, open a terminal and at the proompt, type the following command.

```
% which git
```

The reason for doing this is to determine if **git** is installed on your computer. If so, we can proceed and if not, you will need to install **git** on your compouter or use the computers in the Engineering lab.

Assuming **git** is installed, in the command terminal make a temporary folder using

```
% mkdir gitexample
```

to create a temporary place to work. Then change directories and look at what is in the folder.

```
% cd gitexample
% ls
```

The folder should (but does not need to be) empty. Next, we will initialize a repository in the folder. There are lots of options and flags that can be used with git. We will just use a few in this primer. So, type

```
% git init
```

The command only takes a second or two and will identify the folder as a repository. The output of the command will look something like the following.

```
Initialized empty Git repository in /cygdrive/m/gitexample/.git/
```

Note that the path shown for the folder is dependent on your computer and where you are doing this work.

To see what has been put in the folder you can use the command

```
% ls
```

Unless you have options set, the folder will still look empty. So, instead, type

```
% ls -a
```

to display the hidden files. The command will show a subfolder named **.git** that contains all of the repository bookkeeping for the repository. You never really need to know the contents of this folder and it is highly recommended that the contents are not modified. At least meke sure you know what you are doing in there.

Now, let's put a file in the folder and see what happens in **git**. Type the command

```
% touch hello.f
```

This command creates an empty file that can be modified and worked with. Before modifying the file, type in the **git** command

```
% git status
```

to determine how things are accounted for in the folder. The output from the status command is the following.

```
On branch master

No commits yet

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

        hello.f

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

The output shows that a file is waiting to be included in the repository. To include the file created.

To add a file to the repository, the **git** add and commit command are used to do the work. That is,

```
% git add hello.f

% git commit -a
```

The -a tag is used to include all commits that are listed. The commit command has lots of options for being selective in how to add files and folders. The output looks like the following. During the execution of the commit command, an editor session is started. You must include a comment to the commit to have the command complete the work. All you need is a short comment like

```
% adding hello.f to the repository
```

The output after the editor comment is entered is the following.

```
[master (root-commit) 6a6297f] aaaaaa
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 hello.f
```

This indicates a single file has been included. In the development of code and documentation the commands above are about all you will need to use **git**. However, there is a lot more to the **git** environment.

For example, your entire repository can be pushed up to Github or other VCS sites. We will come back to this. Let's do one more example before continuing. Create a folder called src in the repository.

```
% mkdir src
```

Next, move the file into the folder.

```
% mv hello.f src
```

Move into the folder using

```
% cd src
```

and then edit the file to do the hello world example. That is, using

```
% vim hello.f
```

and edit in the lines

```
      program main
      print *, "hello world"
      stop
      end
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
```

Save the file and then compile the file. That is,

```
     % gfortran hello.f
```

Note that another file will be created named **a.exe** that can be executed as in earlier lectures. Now that we have done a little work, we can use the status command to see how things have changed. Use

```
     % git status
```

which results in

```
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    ../hello.f

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

        ./

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

The result indicates that we need to add the current folder. So, type

```
     % git add ./
```

Finally, commit the changes using

```
     % git commit -a
```

and adding a comment in the editor as above. Once this is done the work is now committed to the repository. It always is a good idea to use the status command to make sure everything has been included or excluded.

The second part of this lesson involves cloning a repository from Github. There are several steps that need to be taken care of first. There are a couple of configuration parameters that need to be set. First move to a directory