Bash Scripting

In this exercise you will create two scripts that focus on using `for` loops and arguments in bash.

cat -n

The `cat` program is used to "concatenate" files together, but it's often abused to simply spit out a short files so you can look at the contents without using a text editor or a pager like `more` or `less`. Another common task is to use the `-n` flag to "number" the lines of output:

```
$ cat -n files/sonnet-29.txt
     1  Sonnet 29
     2  William Shakespeare
     3
     4  When, in disgrace with fortune and men's eyes,
     5  I all alone beweep my outcast state,
     6  And trouble deaf heaven with my bootless cries,
     7  And look upon myself and curse my fate,
     8  Wishing me like to one more rich in hope,
     9  Featured like him, like him with friends possessed,
    10  Desiring this man's art and that man's scope,
    11  With what I most enjoy contented least;
    12  Yet in these thoughts myself almost despising,
    13  Haply I think on thee, and then my state,
    14  (Like to the lark at break of day arising
    15  From sullen earth) sings hymns at heaven's gate;
    16  For thy sweet love remembered such wealth brings
    17  That then I scorn to change my state with kings.
```

You will create a bash script called "cat-n.sh" that mimics this output. Here are the expectations of your program:

- Your program will expect to receive an argument in `$1`
- If there are no arguments, it should print a "Usage" and exit *with an error code*
- If the argument is not a file, it should notify the user and exit *with an error code*
- It will iterate over the lines in the file and print the line number, a space, and the line of the file
- Your output will differ from regular `cat -n` as I won't expect you to right-align the numbers.

Here is the expected behavior:

```
$ ./cat-n.sh
Usage: cat-n.sh FILE
$ ./cat-n.sh files/
files/ is not a file
$ ./cat-n.sh files/sonnet-29.txt
1 Sonnet 29
2 William Shakespeare
3
4 When, in disgrace with fortune and men's eyes,
5 I all alone beweep my outcast state,
6 And trouble deaf heaven with my bootless cries,
7 And look upon myself and curse my fate,
8 Wishing me like to one more rich in hope,
9 Featured like him, like him with friends possessed,
10 Desiring this man's art and that man's scope,
11 With what I most enjoy contented least;
12 Yet in these thoughts myself almost despising,
13 Haply I think on thee, and then my state,
14 (Like to the lark at break of day arising
15 From sullen earth) sings hymns at heaven's gate;
16 For thy sweet love remembered such wealth brings
17 That then I scorn to change my state with kings.
```

head.sh

Write a bash script that mimics the "head" utility where it will print the first few lines of a file. The script should expect one required argument (the file) and a second optional argument of the number of lines, defaulting to 3.

You will create a bash script called "head.sh" that mimics this output. Here are the expectations of your program:

- Your program will expect to receive an argument in `$1` and maybe a second in `$2`
- If there are no arguments, it should print a "Usage" and exit *with an error code*
- If the first argument is not a file, it should notify the user and exit *with an error code*
- If the second argument is missing, use the value "3"
- Print the number of lines requested by the user by iterating over the lines in the file and exiting the loop appropriately
- Do not use the actual `head` command!

Testing

You have been provided a `Makefile` that will run a test suite. This is what it should look like when all tests are passing:

```
$ make test
python3 -m pytest -v test.py
============================= test session starts ==============================
platform darwin -- Python 3.7.0, pytest-3.8.0, py-1.6.0, pluggy-0.7.1 -- /anaconda3/bin/pyth
cachedir: .pytest_cache
rootdir: /Users/kyclark/work/worked_assignments/02-bash-scripting, inifile:
plugins: remotedata-0.3.0, openfiles-0.3.0, doctestplus-0.1.3, arraydiff-0.2
collected 5 items

test.py::test_exists PASSED                                              [ 20%]
test.py::test_usage PASSED                                               [ 40%]
test.py::test_bad_input PASSED                                           [ 60%]
test.py::test_catn_run PASSED                                            [ 80%]
test.py::test_head_run PASSED                                            [100%]

========================== 5 passed in 0.11 seconds ===========================
```

The first test is that the files exist. Then they are tested with no arguments to see if they produce "usage"-type help messages. Then they are tested with bad input. Then they are tested that they produce the expected output when given good input.

This is the basis of "test-driven design" (TDD). We define a set of tests that describe what working software should do. When the tests pass, the software is done. When you are passing all the tests, you are done!

Commit

Remember that I can't `pull` your work until it's been `push`ed it to GitHub.

```
$ git add head.sh cat-n.sh
$ git commit -m 'homework 2' head.sh cat-n.sh
$ git push
```