Linux and Project Tips

- Pagers
  - These are utilities that allow you do view (potentially) long text files, one screen at a time.
  - Text editors like **nano** *can* be used for viewing text files, but are really not built for that purpose.
  - Learn to use a pager like less
    - **Enter** and **Control-P** → To move forward or backward, respectively, by one *line*
    - **Space** and **Control-B** → Forward or backward by one *screen*
    - **q** → To *quit*

Linux and Project Tips

- Pagers
  - Standard navigation keys (up arrow, down arrow, Home, End, etc.) will also often work!
  - The **`less`** utility has *many* more options and customizations of which you can take advantage.
  - Where to look:
    - http://www.thegeekstuff.com/2010/02/unix-less-command-10-tips-for-effective-navigation/
    - At the command line...

```
man less
less --help | less
```

Linux and Project Tips

- I/O redirection
  - Three I/O streams
    - `Standard input` (file descriptor: **0**)
    - `Standard output` (file descriptor: **1**)
    - `Standard error` (file descriptor: **2**)
  - Examples:
    - **Standard output into file (overwrite): [command] > file.txt**
    - **Standard output into file (append): [command] >> file.txt**
    - **Standard error into file (overwrite): [command] 2> file.txt**

Linux and Project Tips

- More redirection examples:
  - **Standard output into file (overwrite), with standard error into standard output:**

```
[command] > file.txt 2>&1
```

  - **Standard output and error into separate files:**

```
[command] 1> output.txt 2>> error.log
```

- ***Pipes*** – let the *output* of one command be *input* to the next:

```
tail -300 /var/log/auth.log | grep Invalid | less
```

  - This allows for better management of output.
  - It is a form of *redirection* that can be combined with the previous

Linux and Project Tips

- Other tips…
  - Tab completion: Type in part of an identifier, press **TAB** for completion
  - Use the ▲ (up) and ▼ (down) arrows to get to commands in your recent CLI history
  - Remember key combos for command line:
    - **Ctrl+A** → *start* of line
    - **Ctrl+E** → *end* of line
    - **Ctrl+U** → delete everything *before* cursor
    - **Ctrl+K** → delete everything *after* cursor
    - **Ctrl** plus **L**/**R** arrow → move cursor one *word* at a time
  - Do not forget **sudo**, especially when editing files. **nano** will not tell you that you lack permissions until you try to save!

Linux and Project Tips

- Final project tips…
  - Assess project needs…
    - *Beginning* state: End of Project 1
    - *Ending* state: After project script is run
    - This may include making an inventory of *which files* are changed, including their start and end states.
  - Plan for ***incremental*** *development* with ***repeated*** *testing*.
    - Start from a known state – i.e., end of Project 1 – with snapshot.
    - Write a little bit of script, and perform test run. Rinse and repeat until you get that little part working right.
    - This helps you to minimize amount of (potentially confusing) error output, on any particular test run.

Linux and Project Tips

- Final project tips…
  - Design for *elegance*.
    - Formatting should look nice and readable. White space is your friend!
    - Use clear variable names.
    - When your code is doing something less than obvious, add comments.
  - Design for easy *adaptability*.
    - The good thing about scripts, like other code, is that you can use sophisticated coding structures to accomplish a lot in fewer lines…
    - …as well as substantially changing results with only minimal changes to actual code.

Linux and Project Tips

- Easy *adaptability*: Consider the following example. Let's say that we want to fill a text file with times tables for the factors 5 and 6, like so:

```
TABLES

1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6

2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12

3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18

4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24

5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
```

Linux and Project Tips

- *The naïve way*: The code and projected output are tightly coupled

```
echo "TABLES" > times_tables.txt

echo "" >> times_tables.txt

echo "1 * 1 = 1" >> times_tables.txt
echo "1 * 2 = 2" >> times_tables.txt
echo "1 * 3 = 3" >> times_tables.txt
echo "1 * 4 = 4" >> times_tables.txt
echo "1 * 5 = 5" >> times_tables.txt
echo "1 * 6 = 6" >> times_tables.txt

echo "" >> times_tables.txt

echo "2 * 1 = 2" >> times_tables.txt
...
```

Linux and Project Tips

- *The naïve way*: This may produce the output desired, but only for the values in question – **5** and **6**
  - What if you want to use different values?
  - Then, you have to write a ***lot*** more code!
  - Even with copying and pasting, it will still be a lot of work.
  - For this project, what if we suddenly wanted to use a different nomenclature for host names? Would you want to rewrite a lot of code for editing the **`/etc/hosts`** file?
  - Let's look at a better way....

Linux and Project Tips

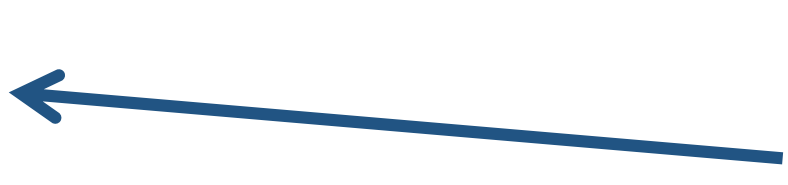
- *A better way*: The code and projected output are de-coupled

```
#! /bin/bash

first=5
second=6

echo "TABLES" > times_tables.txt
echo "" >> times_tables.txt

for i in $(seq 1 $first)
do
    for j in $(seq 1 $second)
    do
        echo "$i * $j = $((i*j))" >> times_tables.txt
    done
    echo >> times_tables.txt
done
```



- If we want to choose numbers *other than* 5 and 6, then we need only change these two lines.
- The rest of the code, *completely unchanged*, will still give the correct output!