1. Scripting
2. Text Editors
3. Customizing
Some Logistics

- (poll) The assignments repository on GitHub.
- Drop deadline is Wednesday 2/10/2016.
Scripting
What is a Script?

• The high-level story is: nothing special.
• Executable filetype.
• Shebang (later).
• Runs from top to bottom.
• The Shebang[5] is used to tell the thing executing the script how (by what program) it should be executed.

• The only time that you technically do not need it is when these two are the same.
  • E.g. you are using a bash shell, and could execute a bash script and be safe.

• You should *always* include the shebang.

• If you are executing using a non-standard program, just include the executable name.
  • Other users may have installed this elsewhere.

• With the shebang, I don't have to do python script.py, I can just do ./script.py.
Execution

- Scripts execute from top to bottom.
- This is just like Python, for those of you who know it already.
- Bad code inside an if statement?
  - You may only realize it when that if statement executes.
Bash Scripting

- Use the shebang: 
  `#!/bin/bash`
- Declare variables...
  ...no spaces!
- Use variables...
  ...dereference with `$`
- Store/use commands executed...
  - `$(command ...)`
  - `command ...`
- If statements and loops.
- NEVER use aliases in bash scripts. EVER.

```bash
#!/bin/bash
#
# declare some variables
NAME="Sven Neys"
MSK_ID=`id -u`
#
# A simple if statement
if [[ $MSK_ID -eq 0 ]]; then
  echo "Executing as root."
else
  echo "Executing as normal user."
fi
#
# A simple string concat
# Note the $ works regardless
echo "You are: $NAME"
#
# A simple for loop using a {} range
for n in {1..11}; do
  echo $n
done
#
# recall that $ needs to be escaped
# with \ to get the actual symbol: \$
```
Caution

- The shebang must be the first line. It must be a valid command.
  - If you expect a custom executable for some reason, then you should only provide the executable name.
    - e.g. `superAwesome` is the executable name, then don't specify the path to your own `superAwesome` executable as the user of the script likely did not install it there.
    - Instead, use `#!/ /usr/bin/env superAwesome`, making the assumption that your user has properly set the their `$PATH` variable to include `superAwesome`.
    - This is different than what I said in lecture, but a much better approach. This is also suggested for how to do it for `python`.

- Not a `#` commentable language?
  - Official answer: just don't use a shebang.
  - Unofficial answer: technically it doesn't matter, since the shebang is a hack on the first 8 bits, but this would render the file useless except for when it is executed by a shell.
VIM and Sublime

- If you have a GUI, I encourage Sublime.
- You do not always get one, so knowing VIM is essential.
  - You are *almost* guaranteed VIM will exist if you don't have a GUI.
- VIM has a LARGE number of shortcuts, you will only learn them with practice.
What is VIM?

- VIM is a powerful "lightweight" text editor.
- VIM actually stands for "Vi IMporoved", where vi is the predecessor.
- VIM can be installed on pretty much every OS these days.
- Allows you to edit things quickly, after the initial learning curve.
The 3 Main Modes of VIM

• Normal Mode:
  • Launching pad to issue commands or go into other modes.
  • Allows you to view the text, but not edit it directly (only through commands).
  • You can jump to normal mode by pressing ESCAPE.

• Visual Mode:
  • Used to highlight text and perform block operations.
  • Enter visual mode from normal mode by pressing v on your keyboard.
    • Visual Line: shift+v
    • Visual Block: ctrl+v
    • Explanation: try them out, move your cursor around...you'll see it.

• Insert Mode:
  • Used to type text into the buffer (file).
  • Like any regular text-editor you've seen before.
  • Enter from normal mode with the i key.
• Most of the time (these days at least), you can scroll with your mouse / trackpad.

• You can also use your arrow keys.

• By design, VIM shortcuts exist to avoid moving your hands at all. Use
  • h to go left.
  • j to go down.
  • k to go up.
  • l to go right.

• With that in mind, the true VIM folk usually map left caps-lock to be **ESCAPE**.
### Useful Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>:help</code></td>
<td>help menu, e.g. specify <code>:help v</code></td>
</tr>
<tr>
<td><code>:u</code></td>
<td>undo</td>
</tr>
<tr>
<td><code>:q</code></td>
<td>exit</td>
</tr>
<tr>
<td><code>:q!</code></td>
<td>exit without saving</td>
</tr>
<tr>
<td><code>:e [filename]</code></td>
<td>open a different file</td>
</tr>
<tr>
<td><code>:syntax [on/off]</code></td>
<td>enable / disable syntax highlighting</td>
</tr>
<tr>
<td><code>:set number</code></td>
<td>turn line numbering on</td>
</tr>
<tr>
<td><code>:set spell</code></td>
<td>turn spell checking on</td>
</tr>
<tr>
<td><code>:sp</code></td>
<td>split screen horizontally</td>
</tr>
<tr>
<td><code>:vsp</code></td>
<td>split screen vertically</td>
</tr>
<tr>
<td><code>&lt;ctrl+w&gt; &lt;w&gt;</code></td>
<td>rotate between split regions</td>
</tr>
<tr>
<td><code>:w</code></td>
<td>save file</td>
</tr>
<tr>
<td><code>:wq</code></td>
<td>save file and exit</td>
</tr>
<tr>
<td><code>&lt;shift&gt;+&lt;z&gt;&lt;z&gt;</code></td>
<td>hold shift and hit z twice: alias for :wq</td>
</tr>
</tbody>
</table>

*Note: `<ctrl+w>` is an alias for `:wq`.*
What?

- VIM is very complicated to start out, but when you memorize the shortcuts it will become crazy fast.
- I suggest you complete the OpenVIM tutorial at [3].
- You can then begin learning the commands, keeping your cheat-sheet[4] handy.
  - Start with lesson 1. When you are ready for more, continue forward.
Customizing
• The `$PS1` variable controls what shows up when you type in your terminal.

• List of all options here:
  
  [link to Bash manual](http://www.gnu.org/software/bash/manual/bashref.html#Controlling-the-Prompt)

• Common: `export PS1="\u@\h:\w> "`
  
  • `usr@hostname:current/working/directory>`

• Try changing your `$PS1` using `export` right now to see how you can modify it.

• Play with colors after, since they are tedious to type in the format needed.
Modifying your Prompt: Aliases

Creating Aliases

alias <new-name> <old-name>

- Used to create alternative ways of entering things, usually commands.
- e.g. alias ..="cd .." means you can just type .. to go up one directory.
- Think of it as copy-pasting. You type new-name and your terminal pastes old-name.
- Should not ever be used in scripts.

- Usually stored in the ~/.bashrc file, though ~/.bash_aliases is slowly gaining traction.
- Make your own!
Storing Customizations

- There are many such places that people put things, but generally speaking...
- Your `bashrc` should have things like aliases and functions. Limit the `export` calls to just things related to coloring the terminal.
- Your `bash_profile` should contain any special environment variables you need to define.
  - Typically when you are exporting things like `$PATH` or `$LD_LIBRARY_PATH` for something you have installed on your own.
- You should source your `bash_profile` from your `profile`, and you should source your `bashrc` from your `bash_profile`. 
Rapid Prototyping

- You may want to quickly change your $PS1 or something and see what it looks like immediately.

- Open your text editor and make the changes you want to see. Flip back to your terminal.

- To reload changes immediately, use the `source` command (e.g. `source ~/.bashrc`).
  - The `bashrc` is reloaded when you open a new terminal.
  - The `profile` (and therefore `bash_profile`) is reloaded when you log in.

- You can `source` the `bash_profile`, but that will only affect the current terminal. In order for all new terminals to get it, you need to log out and log back in.
Follow the instructions in today's lecture demo: https://github.com/cs2043-sp16/lecture-demos/tree/master/lec06
Previous cornell cs 2043 course slides.


Interactive vim tutorial.
Graphical vi-vim cheat sheet and tutorial. 

Shebang (unix).  