PATH Variables

I'm not sure how one first gets exposed to path variables, only that it can be confusing at first glance. This document tries to remedy that without going into the history of **everything**.

The linux file system

What you need to know:

There are various directories that follow this pattern **[not limited to]**:

- `bin`: binaries (actual commands you execute: e.g. `git` and `ssh`)
- `lib`: libraries
- `include`: source code
- `share`: other, usually documentation
- `man`: manual pages

The directories can exist inside of any directory path. There are the usual ones you may have seen already:

- `/bin`
- `/usr/bin`
- `/usr/local/bin`
- `/lib`
- `/usr/lib`
- `/usr/local/lib`

Path Search

Ever wonder what happens when you type in a command (`ssh`) into the terminal and somehow your shell knows to launch that program.

Assuming the command is not a shell command, the shell will check the appropriate variable which includes search locations to launch the program you wish to execute.

In our example, `ssh` is located in `/usr/bin` on my machine.

We can find out where it by using the command `which` which (hehe!) will traverse our search paths to find the executable we are looking for.

```
user@linux1:~$ which ssh
/usr/bin/ssh
```

Intro the the PATH variable

Generally your search path for binaries will look something like the below.

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

We can see what is in our path variable by using the command `echo` to check the contents of the variable that contains our search paths. This variable is called PATH.

```
user@linux1:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

When a search occurs it will take the left most path, search that directory, then proceed to the next path to its right IF and ONLY IF it does not find what it is looking for in the directory it is currently searching.

Fore example: Lets take a look at the executable `pip`.

`/usr/local/bin` is first in the search path so we'll see the version of `pip` installed by CS Techstaff.

```
user@linux1:~$ which pip
/usr/local/bin/pip
```

There also exists and pip executable in `/usr/bin/pip` but as discussed above the shell will use the binary first in the search path.

Setting up an Example

Your shell will allow you manipulate the order in which these paths appear. It's as simple as setting a variable.

For the following examples I assume you are using the bash shell, which is the default in CS. The concept will translate to other shells, though the syntax may be different.

A common example to manipulate your path is if you have written a script and you'd like to invoke it without providing the full path:

Lets create the directory `bin` in the root of our home directory:

```
user@linux1:~$ mkdir ~/bin
```

We will place our example script that echo's 'Hello World!' into `~/bin/hello`

```
#!/bin/bash
echo 'Hello World!'
```

and make it executable.

```
user@linux1:~$ chmod +x ~/bin/hello
```

Now that we have our executable `hello` we'd like to be able to call it without prepending the path like this: `~/bin/hello`

As you may have guessed we will want to add (pre or postpend) `~/bin` to our PATH variable.

Manipulating the PATH variable

```
$ export PATH=$PATH:~/bin
```

To test we will need the shell to reevaluate $PATH.

```
$ hash -r
```

Now we can see that `hello` is in our path and we can execute it without the full path.

```
user@linux1:~$ which hello
/home/user/bin/hello

user@linux1:~$ hello
Hello World!
```

Making the change permanent

Add the following line to a file called `~/.bash_profile`.

```
 export PATH=$PATH:$HOME/bin
```

This will postpend our custom path to the search path every time we launch a bash login shell.

If you'd like to prepend just move your path in front of $PATH. Just make sure each path is separated by a colon `:`.

```
  export PATH=$HOME/bin:$PATH
```