What are Environment Modules?

A typical workstation is used by one or a few users who need a small selection of software packages configured in a specific way. All software is installed under Program Files (Windows), or Applications (Mac), or `/usr/bin` and `/usr/lib` (Linux). Keeping software up to date whilst managing dependencies between specific versions of different software packages is already challenging.

A typical HPC cluster has a large number of users, each needing a different selection of software packages, often with different versions and configurations. Installing all software in `/usr/bin` and `/usr/lib` whilst meeting the disparate needs of each user under these circumstances is simply not possible.

Environment Modules is a tool for managing multiple versions and configurations of software packages, and is used by many HPC centers around the world.

To understand how Environment Modules work, it is helpful to think about what the shell does when you enter a command:

How does the shell know what 'ls' means? (click to expand)

A significant component of the shell is its environment - a set of shell variables and environment variables (such as `$USER`) that scripts and programs can set and use.

A variable has a name, which can contain letters, numbers and underscores, and a value which is simply a text string. To access a variable place a `$` in front of it. Try:

```
echo $USER
```

and compare the result with:

```
echo USER
```

Sometimes you will see a variable reference like `${USER}` or `${USER:-abc}`. The first is a more explicit usage which is needed in certain cases, and the second is a more advanced usage, in this case "if it is set, or abc otherwise". To learn more about advanced usage of variables (also called parameters), type "man bash".

There is a subtle difference between shell variables and environment variables: shell variables are only visible in the current shell, while environment variables are visible to programs started from the shell, including subshells. By convention, shell variables are usually given a lowercase name while environment variables are given an uppercase name.

You can set a shell variable by making it equal something:

```
my_var="hello there"
```

An environment variable is a shell variable, exported to the environment:

```
export MY_VAR="hello there"
```

**Spaces are important!** There must be no spaces on either side of the equal sign.

You can see what environment variables are set with "env".

Environment variables are especially useful when writing job scripts: you can set locations or other options once at the top of the script and reference them later. Changing a run directory or where an input file is kept then becomes much more manageable.

One particularly important environment variable is `$PATH`. This is a colon-separated list of locations in which the shell looks for commands:
$ echo $PATH
/usr/kerberos/bin:/usr/java/latest/bin:/usr/local/bin:/bin:/usr/bin:/opt/ganglia/bin:
:/opt/rocks/bin:/opt/rocks/sbin:/opt/dell/srvadmin/bin:/opt/torque/bin:/opt/torque/sbin:
/home/ab123/bin

So when I enter "ls" at the command prompt, the shell looks for an executable file in /usr/kerberos/bin, then in /usr/java/latest/bin, and so on until it finds one.

This behavior becomes more significant in session 3, when we start to use Environment Modules to make software packages accessible. One of the things that loading an Environment Module does is to add the appropriate directories to your $PATH variable.

To see which executable will be run when you type a command, there is a command called `which`. For example: "which ls" will (probably) show you that entering "ls" at the command prompt will run /bin/ls

With Environment Modules, software packages are installed away from the base system directories, and for each package an associated modulefile describes what must be altered in a user's shell environment - such as the $PATH environment variable - in order to use the software package. The modulefile also describes dependencies and conflicts between this software package and other packages and versions.

To use a given software package, you load the corresponding module. Unloading the module afterwards cleanly undoes the changes that loading the module made to your environment, thus freeing you to use other software packages that might have conflicted with the first one.