Live coding demo - week 2

Demonstration 1: .bashrc

The following commands will make a backup of all dotfiles and then demonstrate that .bashrc is run when a new local terminal is started.

Note: on Mac, .bash_profile or .profile is used instead as the terminal is a login shell.

```
alan@vbox:~$ # Move to your home directory
alan@vbox:~$ cd ~
alan@vbox:~$ # Make a directory to backup your dotfiles
alan@vbox:~$ mkdir Dotfiles/
alan@vbox:~$ # Copy all files starting with . to Dotfiles/
alan@vbox:~$ cp .* Dotfiles/
alan@vbox:~$ # See disk usage of Dotfiles dir
alan@vbox:~$ # this backup costs nothing in space
alan@vbox:~$ du -h Dotfiles/
72K Dotfiles/
alan@vbox:~$ # Make another backup of bashrc to be extra safe
alan@vbox:~$ mv .bashrc .bashrc.bak
alan@vbox:~$ # Add echo command to new .bashrc
alan@vbox:~$ echo 'echo "this is bashrc"' > .bashrc
alan@vbox:~$ cat .bashrc
echo "this is bashrc"
```

Then open a new terminal and you will see the following

```
this is bashrc
alan@vbox:~$
```

As you see, before you have your prompt, the .bashrc was executed and the `echo` command within ran.

If your prompt usually has color, the prompt you now see likely does not. This is because your .bashrc file includes many settings that affect your terminal's appearance and behavior.

These settings are set by storing them in special system variables. In addition, you can set your own variables in .bashrc which will then be avialable in all future sessions.

```
alan@vbox:~$ # Add a variable to the .bashrc file
alan@vbox:~$ echo 'newvar="this is new"' > .bashrc
alan@vbox:~$ cat .bashrc
newvar="this is new"
alan@vbox:~$ # The variable is not available within our current
alan@vbox:~$ # session as .bashrc is only run on startup
alan@vbox:~$ echo $newvar
```

If we now open a new terminal session we see that the `$newvar` variable is set within our environment. This is because the .bashrc file was run when we started the new terminal session.

```
alan@vbox:~$ echo $newvar
this is new
```

Don't forget to put your original .bashrc back!!!

```
alan@vbox:~$ mv .bashrc.bak .bashrc
```

You can keep the backup directory we made in case you ever need to recover one of your dotfiles. As we saw using `du` the directory is small.

Demonstration 2: `export` and `source`

The following commands will use variables to illustrate how `export` and `source` are used to change the scope of variables (i.e., in which environments a variable exists).

`export` sets variables within the scope of your current shell and all subshells.

```
alan@vbox:~$ # Create a variable
alan@vbox:~$ foo=hello
alan@vbox:~$ # The variable is set in our current environment
alan@vbox:~$ echo $foo
hello
alan@vbox:~$ # Start a Bash subshell
alan@vbox:~$ bash
alan@vbox:~$ # In the subshell we just started, the variable does not exist
alan@vbox:~$ echo $foo

alan@vbox:~$ # leave the subshell and return to the parent shell we started in
alan@vbox:~$ exit
exit
alan@vbox:~$ # In this environment the variable is still set
alan@vbox:~$ echo $foo
hello
alan@vbox:~$ # Delete the variable so we can remake it using export
alan@vbox:~$ unset foo
alan@vbox:~$ echo $foo

alan@vbox:~$ # export creates a variable that behaves the same in our current
environment
alan@vbox:~$ export foo=howdy
alan@vbox:~$ echo $foo
howdy
alan@vbox:~$ # Start a subshell again
alan@vbox:~$ bash
alan@vbox:~$ # This subshell has the variable because the parent shell exported it
```

```
alan@vbox:~$ echo $foo
howdy
```

`source` loads variables into your current environment from a file. It does this by executing the commands in the file using your current shell. Essentially it is the same as if you typed the commands out yourself.

```
alan@vbox:~$ # Put a variable assignment command into a file
alan@vbox:~$ echo 'bar=something' > some_file.txt
alan@vbox:~$ # The variable is not set in our current environment as we did not
execute the command in the file
alan@vbox:~$ echo $bar

alan@vbox:~$ # The command is contained in the file
alan@vbox:~$ cat some_file.txt
bar=something
alan@vbox:~$ # If we run source on the file it executes the command, adding the
variable to our environment
alan@vbox:~$ source some_file.txt
alan@vbox:~$ # The variable is now set in our environment
alan@vbox:~$ echo $bar
something
alan@vbox:~$ # Delete the variable again to demonstrate another method
alan@vbox:~$ unset bar
alan@vbox:~$ echo $bar

alan@vbox:~$ # This is the same as the source command above
alan@vbox:~$ # You will see this syntax in places like .bashrc
alan@vbox:~$ . some_file.txt
alan@vbox:~$ # Again, the variable has been set in our environment
alan@vbox:~$ echo $bar
something
```

Demonstration 3: `mamba`

Mamba can be installed following the instructions on the Miniforge Github page https://github.com/conda-forge/miniforge#install

Installation steps:

1. Download shell script
2. Run shell script
3. Hit enter to scroll through license agreement (don't hold enter as it will abort if you press enter after the agreement has ended)
4. Agree to the license agreement by typing `yes` and hitting enter
5. Accept the default settings by typing `yes` when a question offers a choice (indicated with e.g., "[yes|no]") or hitting enter when it suggests a default path
6. Once finished, open a new terminal session or `source` your ~/.bashrc file to activate `mamba` installation. You should see "(base)" appear at the start of your prompt as shown in below code snippet.

Once mamba is installed, you can start creating and using environments. Below is some example usage demonstrated in class

```
(base) alan@vbox:~$ # Having installed mamba, some dirs were added to our $PATH
(base) alan@vbox:~$ echo $PATH
/home/alan/mambaforge/bin:/home/alan/mambaforge/condabin:/usr/local/sbin:/usr/local
/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin
(base) alan@vbox:~$ # Create an environment named "seqtk"
(base) alan@vbox:~$ # and install seqtk from the bioconda channel
(base) alan@vbox:~$ # Specifying a channel with -c adds that channel
(base) alan@vbox:~$ # to the list of channels in your ~/.condarc file
(base) alan@vbox:~$ # All channels in the ~/.condarc and declared using -c
(base) alan@vbox:~$ # will be searched for specified packages and dependencies
(base) alan@vbox:~$ mamba create -n seqtk -c bioconda seqtk
(base) alan@vbox:~$ # mamba will resolve version restrictions of all dependencies
(base) alan@vbox:~$ # and then ask you to confirm the installation with "y"
(base) alan@vbox:~$ # Enter an environment using mamba activate <env>
(base) alan@vbox:~$ # Note "(base)" changes to the environment name
(base) alan@vbox:~$ mamba activate seqtk
(base) alan@vbox:~$ # Activating the environment adds the bin dir for that
(base) alan@vbox:~$ # environment to our $PATH (compare first 2 dirs here to above)
(seqtk) alan@vbox:~$ echo $PATH
/home/alan/mambaforge/envs/seqtk/bin:/home/alan/mambaforge/condabin:/usr/local/sbin
:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
:/snap/bin
(seqtk) alan@vbox:~$ # seqtk is installed in the env-specific bin dir
(seqtk) alan@vbox:~$ which seqtk
/home/alan/mambaforge/envs/seqtk/bin/seqtk
(base) alan@vbox:~$ # Leave the environment and the env bin dir is removed from our
path
(seqtk) alan@vbox:~$ mamba deactivate
(base) alan@vbox:~$ echo $PATH
/home/alan/mambaforge/bin:/home/alan/mambaforge/condabin:/usr/local/sbin:/usr/local
/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin
(base) alan@vbox:~$ # seqtk is no longer callable as its dir isn't on our $PATH
(base) alan@vbox:~$ which seqtk
```