README.md

Class 3 – Introduction to Great Lakes and HPC

Goal

- In this module, we will learn how to work with files and directories using Data Carpentry lesson - Working with Files and Directories
- We will set up our compute environment for upcoming lab modules so that we have all the tools installed and ready for use.
- We will learn how to load Great lakes modules i.e pre-installed softwares provided by ARC-TS team.
- We will learn how to generate a SLURM script and submit our first job to the HPC cluster.

Directory organization for the course

In the first class on Great Lakes we worked in our home directory and copied files over there. For the remainder of the course we will be working in our course home directories. The reason for this is that your home directory is only accessible to you. However, we want to be able to share files, work in teams and have instructors review your work. To accomplish this we have worked with the administrators to setup this course directory with the neccesary permissions to do all of this! To go to this directory use the following cd command (and don't forget about your tab completes!):

```
#Go to the class directory
cd /scratch/epid582w22_class_root/epid582w22_class/

#Check out what's here
ls
```

You will notice that each of you should have your own home directory where you will do work for the class, and also complete your assignments. ***You will also notice that there is a shared_data directory, where bioinformatics programs and data for the course live.***

One final thing which you might ask yourself is how is it that certain users can access certain directories/files, while others can't? For instance, you can access your own course home directory, but not that of your classmates. Similarly, you can all access the shared_data directory, but users not in the class can't. Well, an important thing to be aware of when working in a Unix environment is that users who create files and directories have very tight control over who can read, write or execute the files or within the directories. If you try to go somewhere you don't have permission or perform an operation on a file that you don't have permissions for, then you will be a permission denied error.

So, how do you know what the permissions are and how do you alter them? To examine the permissions we can provide the '-l' flag to ls, which stands for list.

The '-l' flag provides an expanded listing, which provides information on who owns a file/directory and what it's permissions are. Permissions are organized into three blocks indicating read, write and execute permissions for the owner, the group member and everyone else. For instance, the following permission string would be read as follows:

`rwxrw-r--`

1. rwx : The owner can read, write or execute
2. rw- : Group members can read or write, but not execute
3. r-- : Everyone else can read, but not write or execute

For any file you create, you have the power to change permissions using the 'chmod' command. For example, if we wanted to change permission so the user can do anything, the group members can read only and others can't do anything, then we could run the following command

```
chmod u=rwx,g=r,o=- example_file
```

Setting up your compute environment

Setting up environment variables in .bashrc file so your environment is all set for genomic analysis!

Environment variables are the variables/values that describe the environment in which programs run in. All the programs and scripts on your unix system use these variables for extracting information such as:

- What is my current working directory?,
- Where are temporary files stored?,
- Where are perl/python libraries?,
- Where are my tools installed? etc.

In addition to environment variables that are set up by system administators, each user can set their own environment variables to customize their experience. This may sound like something super advanced that isn't relevant to beginners, but that's not true!

Some examples of ways that we will use environment variables in the class are:

1. Create shortcuts for directories that you frequently go to,

2. Setup a conda environment to install all the required tools and have them available in your environment

3. Setup a shortcut for getting on a cluster node, so that you don't have to write out the full command each time.

As an example, let's look at one of the most important environment variables, called the 'PATH' variable. The PATH environment variable contains a list of directories where Unix will look whenever you execute a program or run a command. For instance, when you do something as simple as type 'pwd', the operating system looks in the directories listed in your PATH variable to see if any of them contain a program named 'pwd'. By default, your PATH is setup to include directories where all the Unix commands we have learned live. During the course we will be using some bioinformatics programs that we have performed custom installs on, so we will need to add the directories where we have installed these to the PATH. We will see in a moment how to change an environment variable, but let's quickly see how to see what's in one. To examine the contents of an environment variable we use the 'echo' command. For example, to look at your PATH variable type the following:

```
echo $PATH
```

You will see that PATH contains a colon separated list of paths. So, when you execute a command, Unix goes in order through this list to see if the program exists. We will look at the PATH variable again after we have added to it!

One way to set your environment variables would be to manually set up these variables everytime you log in, but this would be extremely tedious and inefficient. So, Unix has setup a way around this, which is to put your environment variable assignments in special files called .bashrc or .bash_profile. Every user has one or both of these files in their home directory, and what's special about them is that the commands in them are executed every time you login. So, if you simply set your environmental variable assignments in one of these files, your environment will be setup just the way you want it each time you login!

All the softwares/tools that we need in this workshop are installed in a directory

```
/scratch/epid582w22_class_root/epid582w22_class/shared_data/bin
```

and we want the shell to look for these installed tools in this directory.

For this, We will save the full path to these tools in an our PATH variable.

i. Make a backup copy of bashrc file in case something goes wrong.

```
cp ~/.bashrc ~/bashrc_backup_2022_01_12

#Note: "~/" represents your home directory. On great lakes, this means /home/username
```

ii. Open ~/.bashrc file using any text editor and add the following lines at the end of your .bashrc file.

Note: Replace "username" under alias shortcuts with your own umich "uniqname".

There are many different programs that can be used to edit files in a Unix environment. Some of them are extremely complex and powerful, others are simple and there are even options that have a graphical interface for those of you craving that :). In class we will use the most basic text editor available, which is called nano. To edit the .bashrc file, type the follow command.

```
nano ~/.bashrc
```

Once you are in nano you can immediately type and edit. Once you are ready to save and exit, do the following:

1. Save - type ctl-o and then enter
2. Exit - type ctl-x

Now, let's paste the below code block at the end of your .bashrc:

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions

# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/sw/arcts/centos7/python3.8-anaconda/2021.05/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
else
    if [ -f "/sw/arcts/centos7/python3.8-anaconda/2021.05/etc/profile.d/conda.sh" ]; then
        . "/sw/arcts/centos7/python3.8-anaconda/2021.05/etc/profile.d/conda.sh"
    else
        export PATH="/sw/arcts/centos7/python3.8-anaconda/2021.05/bin:$PATH"
    fi
fi
unset __conda_setup
# <<< conda initialize <<<

##epid582 ENV

#Aliases
alias islurm='srun --account=epid582w22_class --nodes=1 --ntasks-per-node=1 --mem-per-cpu=5GB --cpus-per-task=1 --time
alias wd='cd /scratch/epid582w22_class_root/epid582w22_class/username/'

#Great Lakes Modules. They should remain commented until ready for use.
#module load Bioinformatics
```

```
#module load perl-modules

#Bioinformatics Tools installed outside the conda environment due to dependency conflicts.
export PATH=$PATH:/scratch/epid582w22_class_root/epid582w22_class/shared_data/bin/quast-5.0.2
export PATH=$PATH:/scratch/epid582w22_class_root/epid582w22_class/shared_data/bin/multiqc/bin
```

Note: Replace the word "username" under alias shortcuts with your own umich "uniqname".

In the text editor, nano, you can do this by

- Press Ctrl key and "\" key. Nano will then prompt you to type in your search string (here, username).
- Press "return/Enter" key. Then you will be prompted to enter what you want to replace "username" with (here, your uniqname).
- Press "return/Enter" key. Then press "a" key to replace all incidences of username or "y" key to accept each incidence one by one.

You can also customize the alias name such as wd and islurm catering to your own need and convenience.

The above environment settings will set various shortcuts such as "islurm" for entering interactive great lakes session, "wd" to navigate to your workshop directory, call necessary great lakes modules and perl libraries required by certain tools and finally sets the path for bioinformatics programs that we will run during the class.

iii. Save the file and Source .bashrc file to make these changes permanent.

```
source ~/.bashrc
```

iv. Check if the $PATH environment variable is updated

```
echo $PATH

#You will see a long list of paths that has been added to your $PATH variable

# Then test your wd shortcut

wd
```

You should be in your class working directory that is /scratch/micro612w21_class_root/micro612w21_class/username

v. Set up a conda environment using a YML file (you will do this on your own for HW, as it takes a few minutes)

The YML file - `MICRO582_class4_QC.yml` required for generating a conda environment for our next class is located here:

```
/scratch/epid582w22_class_root/epid582w22_class/shared_data/conda_envs
```

Load great lakes anaconda package and set up a conda environment in the following way -

```
# Load anaconda package from great lakes

module load python3.8-anaconda/2021.05

# Set channel_priority to false so that it can install packages as per the YML file and not from loaded channels.

conda config --set channel_priority false

# Create a new conda environment - micro612 from a YML file
```

```
conda env create -f /scratch/epid582w22_class_root/epid582w22_class/shared_data/conda_envs/MICRO582_class4_QC.yml -n M

# Lets check the list of conda environments. You should see the conda environment name on left and the path to the dir

conda env list
```

Loading modules

Great Lakes also provide support for the installation of Bioinformatics software which can be accessed by loading Bioinformatics modules.

```
module load Bioinformatics
```

To check which tools are available under the Bioinformatics module,

```
module av
```

Use the space key to explore the entire suite of tools that are available from Great Lakes. Users can load any of these tools using `module load toolname` command and will be ready for use without the need to install them.

Copy over files for today's lesson

We will again use this data carpentry material to learn how to work with files and directories.

Data for this lesson is located here - `/scratch/epid582w22_class_root/epid582w22_class/shared_data/data/class3`

Change your current location to your working directory and copy class3 folder to your working directory.

```
cd /scratch/epid582w22_class_root/epid582w22_class/username

#OR

wd

cp -r /scratch/epid582w22_class_root/epid582w22_class/shared_data/data/class3 ./

```

Change directory to class3

```
cd class3
```

Submit a job to cluster

Great lakes supports SLURM batch scheduler and resource manager that allows us to run a job on University of Michigan's high performance computing (HPC) clusters.

The Slurm Workload Manager is a job scheduler that gives you access to the HPC nodes/clusters for a specified duration of time so that we can run our memory and data intensive tasks in background. It provides a framework for starting, executing, and monitoring our cluster jobs and can be set in SLURM scripts using the SLURM directives. SLURM directives may appear as header lines in a batch script here first_job.sbat or as options with the the sbatch command line. They specify the resource requirements of your job and various other attributes.

A slurm directive line starts with a "#SBATCH " sign, which lets the bash know that they are header lines and are not a part of code. When we submit the job with a ".sbat" script, the SLURM Workload Manager will read these directives and set/allocates the appropriate clusters for your job.

Some of the key SLURM directives that we recommend using are:

--job-name: This directive sets a job name and makes job monitoring easier when you have multiple jobs running simultaneously on the cluster.

--mail-type: This directive lets you set the email alert notifications for one or all of these events - BEGIN, END, FAIL, ABORT

--mail-user: The email ID to send the mail alert notifications.

--nodes: Number of compute nodes to be assigned for the job.

--cpus-per-task: Number of CPUs to be allocated for the job.

--mem-per-cpu: Minimum memory per CPU processor

--time: Maximum number of days/hours/minutes that the job can run. If the job does not finish within this time frame, it gets killed by the resource manager.

--account: Account to charge the bill and acces the resources assigned under this account

--partition: Request a specific partition for resource allocation instead of let the batch system assign a default partition. If your job requires large memory cores or GPU cores, this is the directive that lets you allocate them for your job.

OK - let's take a look at our first cluster job to see what info is in it:

```
nano first_job.sbat
```

Now, let's submit the job and see what happens!

```
sbatch first_job.sbat
```

Once you've submitted a job it would be nice to be able to check it's status (e.g. is it running? did it error out? is it done?). To do that you use the squeue command, and supply it with your username to get just your jobs:

```
squeue -u username
```

The first column JOBID will give you the job id that was assigned. By default, SLURM manager will log the job in a file "slurm-JOBID.out" and we will explore that file to see what the job did.

OK - now let's look at the output of our job.Make sure to replace the JOBID with the job id that was assigned to your job.

```
less slurm-JOBID.out
```

This website provides a great detailed overview of the process for submitting and running jobs under the Slurm Workload Manager on Great Lakes cluster.

Submit our fasta counter job to the cluster

In the previous class we wrote a nice piece of code that went through the fasta files in a directory and counted the number of sequences in each file. When we ran that code in class, we were executing it on the login node. However, if the files were huge and numerous, we might want to do it on a cluster node instead because:

1. It's frowned upon to run large jobs on the login node, as it slows down things for other users (and ourselves)
2. The cluster is comprised of computers with massive amounts of memory, so an intensive job will run faster if we farm it out to a high memory node
3. If we had a lot of files that we wanted to count the number of sequences in, in principle we could split the job up and run it on multiple nodes. Essentially parallelize our work to get it done much faster! This is in fact the most important and valuable aspect of having access to a compute cluster (- think about processing 1000 genomes in parallel on 1000 compute nodes, versus serially on 1)

OK - so let's open up fasta_counter.sbat in nano to take a look at it, and change the email address.

```
nano fasta_counter.sbat
```

Now, what if we want to submit this cluster job, but run it on a different directory? We, could just edit the sbat file, and resubmit it, but then we've lost our record of the job. So, instead, let's create a new version, edit it and submit that! To create a new version we will use the 'cp' command, which stands for copy.

```
cp fasta_counter.sbat fasta_counter_2.sbat
```

You can type 'ls' to verify that the original and new file exist. Now, let's say that you don't like the name of the file, and want to rename it. To do that you can use the 'mv' command, which stands for move.

```
mv fasta_counter_2.sbat fasta_counter_assembly_2.sbat
```

Now if you type ls, you'll notice that 'fasta_counter_2.sbat' is no longer there! Note that the 'mv' command can be used to rename, but also move files to different directories.

Lastly, edit fasta_counter_assembly_2.sbat to work on the more_fasta directory, and submit it to the cluster!

Turning our fasta counter code into a shell script

Above we applied our for loop to count the number of sequences in a fasta file to two different directories by copying over the code and putting it in a new file. This worked, but is actually bad practice for a few reasons:

1. If you come up with a way to improve your loop (e.g. provide some more informative print outs), you'd have to change it in all the copies you made
2. Even worse, if you find a bug in your code, you have to fix the bug in all copies, which can be tedious and error prone

To avoid copying over our code, we can instead store the code in a file called a shell script, that we can then call everytime we want to run the code!

Shell scripts are just the Unix version of a computer program, and are comprised of sets of Unix commands.

So, first let's put our for loop into a file called 'fasta_counter.sh'.

Use nano to open the file.

```
nano fasta_counter.sh
```

Add the below for loop in fasta_counter.sh shell script.

```
for fasta_file in *.fna
do

        #PRINT THE NAME OF THE FILE
        echo $fasta_file;
        #Count the number of sequences in fasta_file
        grep '>' $fasta_file | wc -l;

done
```

Now, this is nice, but as it stands the shell script will only count the number of sequences in the directory in which it is stored. What would be even better is if we could apply it to count the number of sequences in any directory! To do this, we can give our script a command line argument. Command line arguments are additional information that you provide a shell script, which the shell script can then use to run in a slightly different manner. In bash, command line arguments go into special variables with numeric names (e.g. $1 for the first command line arugment, $2 for the second, etc.)

Lets take an example to understand what those parameters stands for:

```
./some_program.sh Argument1 Argument2 Argument3
```

In the above command, we provide three command line Arguments that acts like an input to some_program.sh These command line argument inputs can then be used to inside the scripts in the form of $0, $1, $2 and so on in different ways to run a command or a tool.

For this script $1 would contain "Argument1" , $2 would contain "Argument2" and so on...

Lets try to incorporate a for loop inside the fasta_counter.sh script that uses the first command line argument - i.e directory name and search for *.fna files in that directory and run fasta sequence counting command on each of them.

- Open “fasta_counter.sh” in nano.
- Input the for loop to count the number of sequences in a fasta file
- Add $1 in the appropriate place where the directory placeholder is needed.
- Run this script on the more_fasta directory and verify that you get the correct results.
- Basic usage of the script will be:

```
./fasta_counter.sh more_fasta
```

You might get an error saying Permission denied. What's the reason? How do we check the permission and set appropriate permission for this file?

▶ Solution