This is a guide on how to set up on the ARC server for your own custom R scripts.

It is suggested to have some knowledge on using Unix command line interface (CLI). There are many many guides on the internet (e.g. search for "unix command line guide"). A non-exhaustive list of oft-used commands:

- `man <command>` (view the manual of `<command>` , press `q` to exit the manual)
- `ls` (list directory content)
- `cd` (change directory)
- `less` (view text file in a new screen, press `q` to exit the view)
- `cat` (prints the content of text file to the terminal, be careful using this on large files)
- `mv` (move or rename file or directory)
- `cp` (copy file)
- `rm` (remove file)
- `rmdir` (remove empty directory)
- `nano` (text editor, press `ctrl-x` to exit)

https://linuxjourney.com/ also provides quite clear and concise guides on numerous Linux basics:

- command line
- text manipulation via command line
- file permissions
- rsync

Table of contents TOC

Prepare R script

Before going to the server, it is preferable that you have finalized your R script locally with a smaller subset of your data. For demonstration, I have an R script, example-msa.R that takes in a fasta file containing ASV sequences and output a fasta file with those sequences aligned.

You should also have the version of the R and R packages you used for your script. Use `sessionInfo()` to show these information. (Make sure the packages are loaded first via `library()`).

Note that whenever you see texts in angular brackets like below, please change the text to what applies to you and also remove the angular brackets.

```
<texts to change>
```

Setup on Server

Login to ARC

Make sure your VPN is on before logging by running the below command.

```
ssh <username>@arc.ucalgary.ca
```

You will then be prompted to enter your password.

(Optional) Remove previous conda installation

If you have previously installed conda and weren't sure what you were doing, I recommend removing conda and re-installing again by following the below steps.

Remove conda (this might take a few minutes):

```
rm -rf ~/miniconda3
```

Remove conda-related configuration files:

```
rm -rf ~/.condarc ~/.conda
```

`rm -rf ~/directory/` means force removing `~/directory/` and all files within `~/directory/` .

Download and install conda

Download `setup-conda.sh` helper script (to download and install conda) to your home directory (the `~` shorthand below). The `\` at the end of the first means that the following line is the same command and not another separate command.

```
wget https://raw.githubusercontent.com/pcrchen/custom/main/script/setup-conda.sh \
    -O ~/setup-conda.sh
```

Running the helper script will download the conda installer Miniconda and install conda automatically. This process takes around 2-3 minutes.

```
bash ~/setup-conda.sh
```

To check if conda was installed correctly, logout the server (type `exit`) and re-login again. You should see `(base)` beside your prompt (e.g. `(base) [username@arc ~]$`). This means that conda is installed and activated, and that you are currently on the default `base` conda environment. More information on conda environments here.

```
exit # exits the server
ssh <username>@arc.ucalgary.ca
```

Another important thing to check is the priority order of the channels conda will use to install packages from (configured in `~/.condarc` file). Conda channels are where packages are hosted on, and the priority order of the channels is important when you install packages. Although, running the `setup_conda.sh` script should have configured the channels properly for you, it doesn't hurt to double check.

If you run the below command to print the content of `~/.condarc` to terminal,

```
cat ~/.condarc
```

the file should look like this:

```
channels:
  - conda-forge
  - bioconda
  - defaults
```

As you can see, `conda-forge` should be at the top, followed by `bioconda` , and `defaults` at the bottom.

If your conda file does not look like the above, run the below command to copy a correct `.condarc` file to your home directory.

```
wget https://raw.githubusercontent.com/pcrchen/custom/main/script/.condarc -O \
    ~/.condarc
```

Setup conda environment

I recommend the use of conda environments to have better reproducibility and avoid possible issues with conflicting packages in one environment. More information on what conda environments are here.

To create a new conda environment to install your packages in:

```
conda create --name <environment-name>
```

For demonstration, I will be creating an environment called `msa` :

```
conda create --name msa
```

To activate/deactivate a conda environment:

```
conda activate <environment-name> # activate env
conda deactivate                  # deactivate env
```

Install packages in the conda environment

Remember your R and R packages and their versions? These are needed to install them using conda.

For `example-msa.R` script, I need `R` version 3.6.3, `optparse` package version 1.6.6, and `msa` package version 1.20.0. To check if they are available (most packages should be) and what the name the package is called on the conda channels we have, use the `conda search` command:

```
conda search <package-name>

# example
conda search r-base   # this is R itself
conda search optparse
conda search msa
```

R packages will be prefixed with either `r-` or `bioconductor-` . From the conda searches, my packages and their versions are available (if not, use the nearest or latest version available).

To install these packages into a specific conda environment use `conda install --name <environment-name>` command:

```
conda install --name <environment-name> r-base=version r-packagename1=version r-packagename2=version

# example
conda install --name msa r-base=3.6.3 r-optparse=1.6.6 bioconductor-msa=1.20.0
```

The specific conda environment don't need to be activated to install if you use the `--name <environment-name>` flag in `conda install` .

The installation may take several minutes (longer the more packages you are installing).

Transferring files to the server

There are several options to transfer your files to the server (I have listed 4 options below, a-d). Your VPN must be on during this process to access ARC.

For Mac and Windows users, I recommend using CyberDuck (option d) to view files on the server that is similar to a file browser. CyberDuck can also be used to transfer files, but if you are transferring many and/or large files, `rsync` will have less issue. Windows users can also use MobaXterm (option c) to view and transfer files.

a. Download directly from a shared Dropbox folder link

Using the below command will save the content of in the shared Dropbox folder in a compressed .zip file located at `~/<path/sample>.zip` . Note the that when you copy the Dropbox shared folder link, at the end of the link is `?dl=0` ; please remember to change the 0 to 1, i.e. `?dl=1` .

```
curl -L -o ~/<path/sample>.zip https://www.dropbox.com/sh/<folder/link>?dl=1
```

Next we need to unzip the downloaded content. The files inside `~/<path/sample>.zip` will be unzipped into the directory `~/<directory/to/unzip/to/>` .

```
unzip ~/<path/filename>.zip -d ~/<directory/to/unzip/to/>
```

Before running either the `curl` or `unzip` command, the directory you want to put the file(s) in must exist. If it doesn't exist, create it by using `mkdir` :

```
mkdir -p ~/<path/to/directory>
```

b. Use `rsync` to copy files from your local computer to the server

Note that the command below will not work if you are logged onto ARC. You can open another terminal window to run the below command, allowing you to stay logged onto ARC on your previous terminal window. I'm not sure if `rsync` will work on MobaXterm on Windows machines.

```
rsync </path/to/local/directory/containing/your/files/> \
    <username>@arc.ucalgary.ca:<path/to/directory/for/your/files/on/arc/>
```

You will be prompted to enter your ARC account password after entering the above command.

If you would like to transfer the entire content of a directory add a recursive flag `-r` or an archive flag `-a` to the `rsync` command. The `-a` flag is recommended if you would like to preserve timestamp among other things (more information by looking at `rsync` manual using `man rsync` command).

```
rsync -r </path/to/local/directory/containing/your/files/> \
    <username>@arc.ucalgary.ca:<path/to/directory/for/your/files/on/arc/>
# or
rsync -a </path/to/local/directory/containing/your/files/> \
    <username>@arc.ucalgary.ca:<path/to/directory/for/your/files/on/arc/>
```

c. Use MobaXterm for Windows users to transfer

I believe there is an "up arrow" icon on the left panel to upload files. More information available in this guide. Note that this may not be a great option if you are uploading many and/or large files.

d. Use Cyberduck for Windows and Mac users to transfer

Detailed instructions can be found in this guide. The settings for SFTP after clicking "Open Connection" will be different:

- Server: arc.ucalgary.ca
- Username: username for your ARC account
- Password: password for your ARC account
- SSH Private Key: None
- Check "Save Password" if you want

Note that this may not be a great option if you are uploading many and/or large files.

Side note on files structures for this example

I tend to keep consistent file structure for each project. For this `example-msa.R` "pipeline", my files and directories will look like the following:

```
~/msa                   # name of the project
~/msa/data              # contains all raw data for this analysis
~/msa/data/sample_set1/
~/msa/data/sample_set2/
~/msa/script            # scripts for running this analysis
~/msa/slurm             # Slurm scripts for running this analysis
~/msa/out               # contains all output for this analysis
~/msa/out/sample_set1/
~/msa/out/sample_set2/
```

Prepare Slurm batch script

ARC uses the Slurm job scheduler for managing batch script job submission. Please don't run any heavy computational commands on the login node. More information on the Slurm job scheduler here.

Below is an example slurm script suitable for data size up to 6 GB uncompressed (`*.gz` compressed files are around ~4 times smaller).

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --partition=single,lattice
#SBATCH --cpus-per-task=8
#SBATCH --mem=0
#SBATCH --time=48:00:00
#SBATCH --job-name=jobname
#SBATCH --output=slurm-%x-%J.out
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=<username>@ucalgary.ca


# Reads in Bash shell configuration to allow conda activation
source ~/.bashrc

# Activate conda environment
conda activate your_environment

# Commands
Rscript your_R_script.R

# Prints out date and time of completion
date
```

`SBATCH` options:

- `#SBATCH --nodes=1` specifies the number of nodes to use. For our use cases, this should always be 1.
- `#SBATCH --ntasks=1` specifies the number of tasks to use. For our use cases, this should always be 1.
- `#SBATCH --partition=single,lattice` specifies the computing cluster this use on ARC. For more information on the available clusters and their resources see the table below. Here, it is requesting either the `single` or `lattice` cluster. I don't recommend using the GPU partition (`gpu-v100`) if you don't need a GPU for analysis, as GPU partition is in high demand.
- `#SBATCH --cpus-per-task=8` specifies the number of cores to use for multithreaded commands. Make sure you are specifying the same number as in your R script. Here, it is specifying 8 cores (allowing 8 multithreads).
- `#SBTACH --mem=0` specifies the amount of memory to use. The default unit is in MB. Here, with a value of 0, it is specifying to use all available memory on the node (that would be 12 GB for `single` and `lattice` nodes).
- `#SBATCH --time=48:00:00` specifies the time limit for the job in HH:MM:SS format. You will have higher priority in the queue if this time limit value is lower. Here, the time limit is 48 hours. And different cluster partition have different maximum allowed time limit (see table below).
- `#SBATCH --job-name=jobname` specifies the name for this job. Here, the job name is `jobname` .
- `#SBATCH --output=slurm-%x-%J.out` specifies the location to save the terminal output from the job submission. `%x` is the job name and `%J` is the job ID. Because the path doesn't begin with `/` , it is a relative path, and so the output file will be saved relative to the directory you submit the job using `sbatch` . If you want to make sure the output is always saved to the same directory, use an absolute path. e.g. `--output=/home/<username>/slurm/%x-%J.out` .
- `#SBATCH --mail-type=END,FAIL` specifies if you want email notification. Here, an email will be sent when the job completes or fails. If you don't want this, change the value to `NONE` . You can also add another value to email you at the start of the job as well (`BEGIN,END,FAIL`).
- `#SBATCH --mail-user=<username>@ucalgary.ca` specifies the email for the notifications. Here, email

will be sent to `<username>@ucalgary.ca` .

Partition	Cores per node	Memory limit (MB)	Time limit (h)	GPUs pernode
gpu-v100	40	753000	24	2
bigmem	80	3000000	24	
cpu2021	48	185000	168	
cpu2019	40	185000	168	
cpu2013	16	120000	168	
parallel	12	23000	168	
lattice	8	12000	168	
single	8	12000	168	

Note that the above table may be outdated. Refer to the ARC Cluster Guide for up to date information.

You can download this example Slurm batch script to ARC to edit using the command line interface text editor `nano` :

```
wget https://raw.githubusercontent.com/pcrchen/custom/main/slurm/template.slurm \
    -O ~/<path/to/your/desired/location>.slurm

nano ~/<path/to/your/desired/location>.slurm
```

After finishing editing with `nano` , press `ctrl-o` to save. Near the bottom there should be a prompt saying "File Name to Write", either just press `enter` to save to the same file or change the filename and then press `enter` . Once saved, press `ctrl-x` to exit.

More information on how to use `nano` editor here.

Another method is to edit the slurm file locally and then upload it to the server.

Example slurm script for running example-msa.R

Here I will be downloading the Slurm batch script template and edit using `nano` to run the `example-msa.R` script on a computing node.

```
wget https://raw.githubusercontent.com/pcrchen/custom/main/slurm/template.slurm -O \
    ~/msa/slurm/example.slurm

nano ~/msa/slurm/example.slurm
```

My Slurm batch looks like below after editing:

```
#!/bin/bash

#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --partition=single,lattice
#SBATCH --cpus-per-task=8
#SBATCH --mem=0
#SBATCH --time=48:00:00
#SBATCH --job-name=msa-example
#SBATCH --output=/home/<username>/slurm/%x-%J.out
#SBATCH --mail-type=END,FAIL
#SBATCH --mail-user=<username>@ucalgary.ca


# Reads in Bash shell configuration to allow conda activation
source ~/.bashrc

# Activate conda environment
conda activate msa
```

```
# Commands
Rscript ~/msa/script/example-msa.R \
        --input ~/msa/data/example/asv.fasta \
        --output ~/msa/out/example/aligned_asv.fasta

# Prints out date and time of completion
date
```

For `example-msa.R`, I have specifically written it so that I can use command line arguments (i.e. the `--input` and `--output` flags), so that I don't have to edit `example-msa.R` each time I want to have different input and output files. Of course if you just specified the input and output files in your R script, the command under `# Commands` would just be this:

```
# Commands
Rscipt ~/path/name_of_Rscript.R
```

Submit Slurm batch script

With the Slurm batch script edited, run the below command to submit it with `sbatch`:

```
sbatch ~/</path/to/your_slurm_script>.slurm

# example
sbatch ~/msa/slurm/example.slurm
```

Check status of batch script job submission

Using the `squeue` command can show the status of active jobs (either in queue or running) you have submitted.

```
squeue -u <username>
```

An email may also be sent to you depending on your `--mail-type` setting.

Results files

Result files will be saved to the location you specified in the R or Slurm batch script. To transfer the result outputs refer to earlier section on transferring files.

Using conda

Speed up conda installation

Recently, another package called `mamba` was developed that can be used to speed up conda installations. To use mamba, we need to first install it in conda default base environment.

```
conda activate base # to ensure we are in the base environment

conda install mamba
```

To use mamba to install conda packages:

```
mamba <package-name>
```

Or to use mamba to install in new conda environment.

```
mamba create --name <environment-name> <package-name>
```

More information on mamba here.

Conda environments

Conda environments are used to compartmentalize your installed packages, because, for example, one may need to use different pipelines that require different versions of the same package.

You can think of a conda environment as a physical lab, and the different conda environments are the different labs used for different purposes (e.g. molecular lab, parasitology lab, etc.). The equipment available in different labs are the different packages installed in each conda environment. If you activate a particular conda environment, you cannot access the packages installed in other environments, much like you cannot access the equipment in the parasitology lab when you are in the molecular lab.

By default, conda will create the `base` environment for you, and you will be in the `base` conda environment when you first install and activate conda.

You can activate a conda environment by running the following command:

```
conda activate <environment-name>
```

Once activated, `(environment-name)` should replace `(base)` before your command prompt (e.g. `(environment-name) [username@arc ~]$` .

To deactivate the environment:

```
conda deactivate <environment-name>
```

To list the conda environments you have:

```
conda info --env
```

More information on managing conda environments here.

auto activate base

`auto_activate_base` is a configurable setting in conda. The default value is `True` , which means that conda will be automatically activated (shown by the `(base)` beside your command prompt) whenever you login to an account with conda installed (this would be your server account in this case).

To turn this off, run the below command:

```
conda config --set auto_activate_base False
```

I don't recommend turning this setting to `False` . Unless you want to use the module system provided on ARC, because the packages installed in conda base environment may conflict with the packages from the module system. I personally prefer to use conda for managing and installing packages, as it is well-documented and handles package dependencies pretty well.

More information on configuring conda here.

Update conda

This will update only conda itself and not the packages installed via conda.

```
conda update -n base -c defaults conda
```

Other resources for conda

- Conda cheatsheet (very useful)
- Conda documentation
- Installing conda on Linux
- Quick start on using Conda
- Uninstalling Miniconda3 on Linux

ARC Cluster Guides

Below are links to guides provided the UC's Research Computing Services (RCS):

- RCS home page
- ARC cluster guide
- ARC cluster general guidelines and policies
- What is a job scheduler
- Running slurm jobs
- Bioinformatic application example

- Linux Introduction
- How to transfer data
- UC high performance computing (HPC) systems
- HPC Linux topics: A list of topics on which RCS technical support staff can provide one-on-one or group training
- Courses offered by RCS
- Data storage options for UofC researchers
- Security and privacy