

---

# buddy-documentation

**CREIC**

**Jan 11, 2023**



# QUICKSTART

<b>1</b>	<b>About</b>	<b>1</b>
<b>2</b>	<b>Cluster Usage: Rules and Guidelines</b>	<b>3</b>
2.1	General Etiquette . . . . .	3
2.2	Rules . . . . .	3
2.3	Guidelines . . . . .	3
<b>3</b>	<b>Connecting to Buddy</b>	<b>5</b>
3.1	Get an Account . . . . .	5
3.2	Connect Through Open OnDemand . . . . .	5
3.3	Connect Through SSH . . . . .	5
<b>4</b>	<b>Open OnDemand</b>	<b>7</b>
4.1	Access and Login . . . . .	7
4.2	Interactive Applications . . . . .	8
4.3	File Browser . . . . .	10
4.4	File Editor . . . . .	13
4.5	Terminal Access . . . . .	14
4.6	Job Management . . . . .	14
4.7	Job Composer . . . . .	16
<b>5</b>	<b>Using the Terminal</b>	<b>23</b>
5.1	Background . . . . .	23
5.2	Accessing the Terminal . . . . .	23
5.3	Terminal Basics . . . . .	25
5.4	Common Commands and Features . . . . .	30
5.5	Tips and Tricks . . . . .	37
5.6	Basic Bash Scripting . . . . .	38
<b>6</b>	<b>Slurm</b>	<b>43</b>
6.1	Terminology . . . . .	43
6.2	Commands . . . . .	44
6.3	Sbatch Parameters . . . . .	44
<b>7</b>	<b>Common toolchains</b>	<b>45</b>
7.1	Component versions in foss toolchain . . . . .	45
7.2	Component versions in intel toolchain . . . . .	45
7.3	Component versions in foss toolchain (deprecated versions) . . . . .	46
7.4	Component versions in intel toolchain (deprecated versions) . . . . .	46
<b>8</b>	<b>Data Storage</b>	<b>47</b>

8.1	Home Folder . . . . .	47
8.2	Project Space . . . . .	47
8.3	Scratch Storage . . . . .	47
<b>9</b>	<b>Data Transfer</b>	<b>49</b>
9.1	Using SFTP or SCP . . . . .	49
9.2	Using Github . . . . .	49
9.3	Using Globus . . . . .	49
<b>10</b>	<b>Account and Software Requests</b>	<b>51</b>
10.1	General Account Request . . . . .	51
10.2	Classroom Account Request . . . . .	51
10.3	Software Request . . . . .	52
<b>11</b>	<b>Tips and Tricks</b>	<b>53</b>
11.1	Slurm . . . . .	53
11.2	Python . . . . .	54
<b>12</b>	<b>Overview</b>	<b>55</b>
<b>13</b>	<b>ANSYS</b>	<b>57</b>
<b>14</b>	<b>COMSOL</b>	<b>59</b>
<b>15</b>	<b>Anaconda</b>	<b>61</b>
15.1	Example Script . . . . .	61
<b>16</b>	<b>Gaussian</b>	<b>63</b>
16.1	Example Script . . . . .	63
<b>17</b>	<b>Jupyter/Python</b>	<b>65</b>
<b>18</b>	<b>R/RStudio</b>	<b>67</b>
<b>19</b>	<b>Stacks</b>	<b>69</b>
<b>20</b>	<b>Overview</b>	<b>71</b>
<b>21</b>	<b>Advanced Slurm</b>	<b>73</b>
<b>22</b>	<b>Array Jobs</b>	<b>75</b>
<b>23</b>	<b>Using Git and Github</b>	<b>77</b>
<b>24</b>	<b>Machine Learning</b>	<b>79</b>
<b>25</b>	<b>OMPI</b>	<b>81</b>
<b>26</b>	<b>OURRstore</b>	<b>83</b>

## ABOUT

Buddy is a supercomputer managed by [CREIC](#) and funded by the National Science Foundation. The following is a break down of our system

Table 1: Buddy Nodes

Name	Quantity	Node Type	Cores	RAM	Features
buddy.uco.edu	1	Login	2x8	64GB	Open OnDemand
buddy-##	31	Compute	2x10	64GB	Compute
buddy-high-mem-##	4	High Memory	2x10	128GB	High Memory
Buddy-gpu-##	2	GPU	2x10	64GB	Tesla K40
buddy-dtn-##	2	Data Transfer	2x8	64GB	OFFN+Globus
Storage-ib	1	Storage	2x8	64GB	Home/Project Storage

Buddy is being used for both research and education. Research includes projects on machine learning, particle transport, micromixing, stochastic modeling, ecological modeling, bio-informatics, spread of disease, and much more! Buddy also has JupyterLab which is heavily utilized in classroom environments. You can find grant details [on the NSF site](#).



## CLUSTER USAGE: RULES AND GUIDELINES

### 2.1 General Etiquette

The cluster is a shared resource which we hope will be heavily utilized by anyone who could make use of it. We encourage users to take advantage of this resource while using judgment to avoid inconveniencing other users, overburdening the cluster, or abusing it.

### 2.2 Rules

There are a few rules that must be followed to prevent overburdening or abusing the cluster.

#### 2.2.1 1. Do not store sensitive information on the cluster

Buddy is not yet HIPAA compliant. Do not store student grades, individually identifiable health information, or other such sensitive information on the cluster at this time.

#### 2.2.2 2. Do not abuse cluster resources

Buddy is intended to support research and learning. Do not use the cluster to mine cryptocurrency, crack passwords, spam emails, or other such intensive and inappropriate activity.

### 2.3 Guidelines

Here are a few guidelines for being considerate on the cluster. If you have any questions please contact administration.

#### 2.3.1 1. Avoid running scripts directly

Buddy uses slurm to schedule jobs and allocate resources. This allows us to ensure every user has enough resources for their application and everyone has the ability to utilize the cluster. Try to work within slurm to allow us to keep buddy healthy and usable for everyone.

### **2.3.2 2. Avoid running jobs on the head node**

If you are accessing buddy via a terminal and not using slurm (see guideline 1) you will initially be working on the head node (hostname buddy). The head node is meant to manage the cluster while compute nodes handle the workload. Please ssh into a compute node (hostname buddy-01,buddy-02,etc.).

### **2.3.3 3. Use globus to transfer large files onto the cluster**

While it is acceptable to use open ondemand, sftp, or scp to transfer data onto the cluster, if the data is over a certain size these methods can overburden the cluster causing a slowdown for all other users. For large files, be sure to utilize our DTNs and Globus. See the [data transfer](#) section for more information.

### **2.3.4 4. Run jobs in an appropriate partition**

Slurms jobs on Buddy are run within partitions that are optimized for different use cases. For most applications the general partition is appropriate, however, if your job requires a lot of memory use a high-mem partition or if you expect your job will need to run for longer than 5 days use a long partition. See [slurm partitions](#) for more information.

### **2.3.5 5. Avoid allocating unnecessary resources**

Feel free to allocate enough resources for your application but try not to allocate resources you are not going to need. If you require a large quantity of resources use the appropriate partition or contact administration.

### **2.3.6 6. Avoid running unnecessary jobs**

Prefer running multiple tasks within one job over spawning multiple jobs when possible.

### **2.3.7 7. Avoid queueing an extreme number of jobs**

Try to allow enough room in the queue for other users to run their jobs. If you have an application that requires a large number of jobs use the appropriate partition or contact administration.

### **2.3.8 8. Be aware of software licenses**

Some software on buddy requires a software license such as COMSOL. There are only so many licenses which means only a few users can access that software at a time. Keep this in mind and try to give others an opportunity to access the software as well.



## CONNECTING TO BUDDY

### 3.1 Get an Account

Email administration at [hpc@uco.edu](mailto:hpc@uco.edu) and we will create an account for you. Please read the *rules and guidelines* before using Buddy.

### 3.2 Connect Through Open Ondemand

If you prefer to use a browser you can access Buddy through <https://buddy.uco.edu>. This will take you to our Open Ondemand page which makes it easy to setup jobs and use graphical software. See *Open Ondemand* for more information.

### 3.3 Connect Through SSH

If you prefer a terminal you can connect to buddy using ssh. See *Using the Terminal* for more information.



## OPEN ONDEMAND

Open OnDemand is our new and improved interface for accessing HPC resources on Buddy. Open OnDemand is developed and maintained by Ohio Supercomputing Center. You can access it by visiting <https://buddy.uco.edu>. This document describes how to access and use OnDemand and it's services.

UCO Central Oklahoma Files Jobs Clusters Interactive Apps My Interactive Sessions Develop Help Logged in as skelling1 Log Out

**UCO** UNIVERSITY OF CENTRAL OKLAHOMA  
Center for Research  
and Education in  
Interdisciplinary Computation

Message of the Day

April 2022 - User Notice

Maintenance is scheduled for the first and third Thursday of every month. Jobs should continue to run without interruption.

Jupyter now supports custom conda environments! Simply check advanced and change the version to custom.

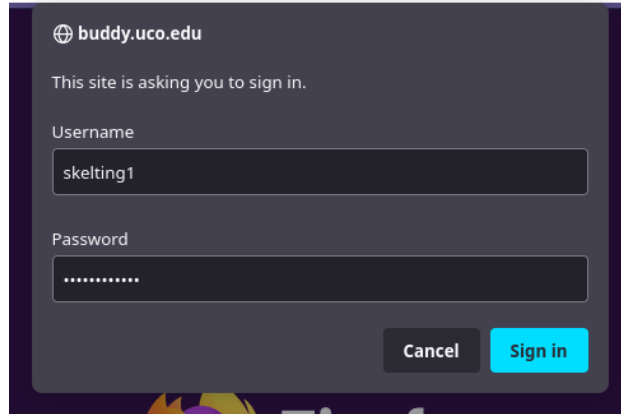
VSCode Server has now been added to Buddy! You can access it through your interactive applications. Please note that it is still in beta, and features such as github are not working. If you notice any issues during use, please email [hpc@uco.edu](mailto:hpc@uco.edu) so we can track them.

We are always adding new features and OnDemand applications! If you have any software requests, please let us know at [hpc@uco.edu](mailto:hpc@uco.edu).

powered by **OPEN OnDemand** OnDemand version: v2.0.20

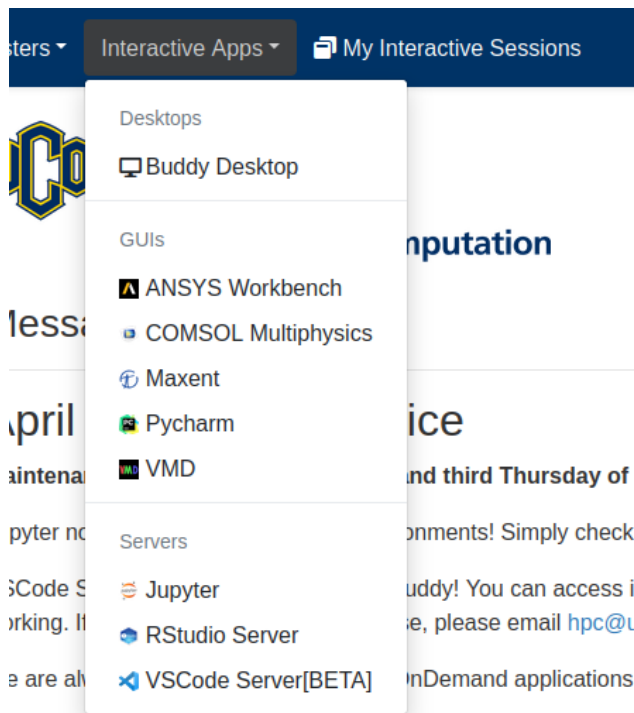
### 4.1 Access and Login

OnDemand can be accessed by visiting <https://buddy.uco.edu>. You will login using the credentials provided by CREIC. If you do not already have an account, or you require a password reset, please email [hpc@uco.edu](mailto:hpc@uco.edu) for assistance. Please note that while the username may match yours at uco, this account is not connected to your UCO credentials.



## 4.2 Interactive Applications

Buddy has a number of interactive applications available for use. You can access them via the interactive apps tab, or by clicking “My Interactive Sessions”



As an example, we will access Jupyter. Below is what the application will most likely look like. We will touch on each of the available options, including the advanced tab.

Interactive Apps

Desktops

Buddy Desktop

GUIs

ANSYS Workbench

COMSOL Multiphysics

Maxent

Pycharm

VMD

Servers

Jupyter

RStudio Server

VSCoDe Server[BETA]

### Jupyter version: v3.1.2

This app will launch a Jupyter Lab or Notebook server on one or more nodes.

Queue

General

Queue your job will run on

Jupyter Session Type

Jupyter Lab

Choose between Jupyter Notebook and Jupyter Lab

Number of hours

2

Set the length of time for this job (1-48)

Number of cores

2

Set the number of cores for this job (1-20)

☒ Data Science Toolkit

☒ View advanced options

Jupyter version

Jupyter 3.0.16 (Python 3.9.5/FOSS 2021a/GCC 10.3.0)

This defines the version of Jupyter you want to load

Additional modules

Additional modules you wish to load seperated by spaces. Please be sure to match the FOSS, GCC, and or Python versions listed with your selected Jupyter version.

Launch

\* The Jupyter session data for this session can be accessed under the [data root directory](#).

### 4.2.1 Queue

Allows you to select the queue your application will run on. General is typical for most jobs. GPU and High Memory are available for specialized work loads, but there are a limited number of nodes with these resources.

### **4.2.2 Number of Hours**

This is the number of hours your interactive application will run for. Please note there is a 48 hour limit on all interactive app jobs. If you require longer runtimes, please utilize a SLURM script.

### **4.2.3 Number of Cores**

Number of cores for your job. You can reserve up to 20, but only use the minimum required. Two is typically best for classroom jobs, and most research jobs. Twenty will reserve the entire node for just that application. This should only be done when absolutely needed.

### **4.2.4 Version**

This selects the version of the application you wish to run. Most often, you will want to pick the latest version, unless you have some need that requires an older revision.

### **4.2.5 Additonal Modules**

Additional modules can be added here. This utilizes LMOD, and will automatically load the user supplied list of modules. Please be sure your modules toolchain version matches the toolchain of your software version. You can read more in the module section about toolchains.

### **4.2.6 Other Options**

Certain applications contain other options. For example, Jupyter let's you choose between a Lab or Notebook session. Jupyter also offers ready to select module groups like "Data Science".

## **4.3 File Browser**

OnDemand offers a built in file browser. You can access it by going to Files>Home Directory. The file browser has options to upload files, edit text files, general file management, and more all within the web browser! Applications like Filezilla are no longer needed to move data to and from Buddy.

Home Directory

/ home / skelling1 / Documents / [Change directory](#) [Copy path](#)

☐ Show Owner/Mode ☐ Show Dotfiles Filter:

Showing 4 rows - 0 rows selected

Type	Name	Size	Modified at
Folder	Data Folder 01	-	4/21/2022 11:37:38 AM
Folder	Data Folder 02	-	4/21/2022 11:37:44 AM
File	batchjob.sh	0 Bytes	4/21/2022 11:38:04 AM
File	slurm_output.txt	0 Bytes	4/21/2022 11:37:55 AM

powered by **OPEN OnDemand** OnDemand version: v2.0.20

### 4.3.1 Top Menu

Most file tasks can be performed via the menu in the upper left.

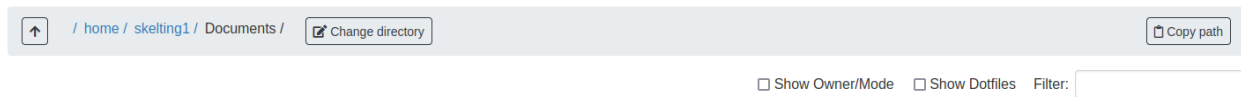


- **Open in Terminal:** Opens your current folder in a terminal via a new tab. The side arrow allows you to select the cluster you want to open the terminal on. At this time, there is only Buddy. So this option is not needed.
- **New File:** Opens a dialogue to create a new file in the current folder
- **New Directory:** Opens a dialogue to create a new folder in the current folder
- **Upload:** Opens a dialogue to upload desired files or folders
- **Download:** Downloads files or folders that have been selected
- **Copy/Move:** Opens a dialogue to copy or move files or folders that have been selected
- **Delete:** Deletes selected files or folders.

**Warning:** File deletion is permanent on Buddy, both in the file browser and terminal! There is no “trash”. In addition, files cannot be recovered due to the nature of Buddy.

### 4.3.2 Navigation Menu

The navigation menu allows has additional navigation options that some users may find useful

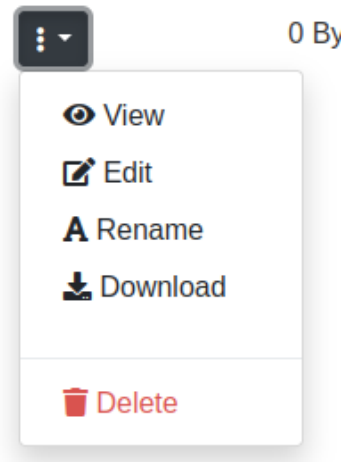


- **Up Arrow:** Goes up a directory
- **Path Bar:** Shows the path to your present working directory
- **Change Directory:** Allows for navigation to a specific folder by providing a path
- **Copy Path:** Copies the path to your present working directory
- **Show Owner/Mode:** Shows file and folder permissions
- **Show dotfiles:** Shows hidden files and folders
- **Filter:** Filter current files and folders by name

### 4.3.3 File Context Menu

The file context menu provides a number of operations that mose users will find useful

slurm\_output.txt

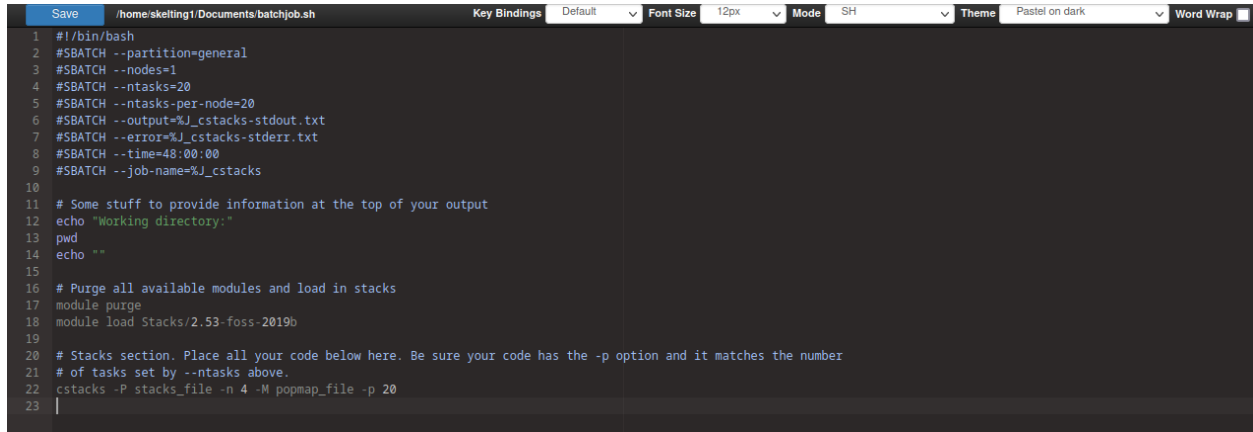


- **View:** Opens text files for viewing in a new window
- **Edit:** Opens files in the OnDemand text editor in a new tab
- **Rename:** Opens a dialogue to rename files
- **Download:** Downloads the file



## 4.4 File Editor

OnDemand has a built in file editor that you can use to modify any text based file. This is perfect for modifying scripts, input data, and a number of other tasks.



The screenshot shows the OnDemand file editor interface. At the top, there is a toolbar with buttons for 'Save', 'Key Bindings', 'Default', 'Font Size' (set to 12px), 'Mode' (set to SH), 'Theme' (set to Pastel on dark), and 'Word Wrap'. Below the toolbar, the editor displays a script file named `batchjob.sh` located at `/home/skelting1/Documents/batchjob.sh`. The script content is as follows:

```

1 #!/bin/bash
2 #SBATCH --partition=general
3 #SBATCH --nodes=1
4 #SBATCH --ntasks=20
5 #SBATCH --ntasks-per-node=20
6 #SBATCH --output=%J_cstacks-stdout.txt
7 #SBATCH --error=%J_cstacks-stderr.txt
8 #SBATCH --time=48:00:00
9 #SBATCH --job-name=%J_cstacks
10
11 # Some stuff to provide information at the top of your output
12 echo "Working directory:"
13 pwd
14 echo ""
15
16 # Purge all available modules and load in stacks
17 module purge
18 module load Stacks/2.53-foss-2019b
19
20 # Stacks section. Place all your code below here. Be sure your code has the -p option and it matches the number
21 # of tasks set by --ntasks above.
22 cstacks -P stacks_file -n 4 -M popmap_file -p 20
23 |

```

There are a number of options available within the editor

### 4.4.1 File Options

- **Save:** Saves your currently open file
- **Path:** Shows the name and location of the file you currently have open

### 4.4.2 Editor Options

- **Key Bindings:** Allows for special key bindings
  - **Default:** This option is best for most users as this is the standard set of key bindings to which desktop users are accustomed.
  - **Vim:** VIM types bindings, including common modes such as command, insert, replace, and block. This mode is not recommended unless you use VIM.
  - **Emacs:** Emacs type bindings. This mode is not recommended unless you use Emacs.
- **Font Size:** Size of font displayed within the editor
- **Mode:** This is typically automatically selected based on your file extension. The mode controls syntax highlighting and can help to discern elements when performing tasks like writing a script.
- **Theme:** Changes how your text editor looks. Both light and dark themes are available.
- **Word Wrap:** Marks whether to wrap words. This prevents having to scroll horizontally for extremely long lines.

## 4.5 Terminal Access

A terminal can be accessed by going to Clusters>Buddy Shell Access. This terminal is web based will open a terminal in a new window. This means that applications like Putty are no longer needed to access Buddy.

```

Host: buddy.uco.edu
Last login: Thu Apr 21 13:23:47 2022 from buddy.uco.edu
## April 2022 - User Notice

**Maintenance is scheduled for the first and third Thursday of every month. Jobs should continue to run without interruption.**

Jupyter now supports custom conda environments! Simply check advanced and change the version to custom.

VSCode Server has now been added to Buddy! You can access it through your interactive applications. Please note that it is still in beta
, and features such as github are not working. If you notice any issues during use, please email hpc@uco.edu so we can track them.

We are always adding new features and OnDemand applications! If you have any software requests, please let us know at hpc@uco.edu.

---
[skelting1@buddy ~]$

```

Much like the text editor, the terminal also has a theme option. To learn more about bash and common Linux commands, see our guide on the terminal and Slurm.

## 4.6 Job Management

Jobs Management can be accessed by going to Jobs>Active Jobs. This allows users to manage their slurm jobs, as well as see jobs from other users running on the cluster. Please see our Slurm section for more information about jobs. Please note that this application is paginated, and you may need to mark to show more entires or click through available pages using the navigation at the bottom.

Central Oklahoma

Files

Jobs

Clusters

Interactive Apps

My Interactive Sessions

Develop

Help

Logged in as skelting1

Log Out

Active Jobs

Job Composer

All Jobs

All Clusters

Active Jobs

Show 50 entries

Filter:

ID	Name	User	Account	Time Used	Queue	Status	Cluster	Actions
19	buddy/sys/dashboard/sys/buddy_jupyter		buddy-users	37:26:26	general	Running	Buddy	
67	buddy/sys/dashboard/sys/buddy_jupyter		buddy-users	02:53:35	general	Running	Buddy	
29	sample_job		buddy-users	00:47:16	general	Running	Buddy	
31	buddy/sys/dashboard/sys/buddy_ansys		buddy-users	00:31:39	general	Running	Buddy	
33	buddy/sys/dashboard/sys/buddy_jupyter		buddy-users	00:20:45	general	Running	Buddy	
34	buddy/sys/dashboard/sys/buddy_ansys		buddy-users	00:10:23	general	Running	Buddy	

Showing 1 to 6 of 6 entries

Previous

1

Next

## 4.6.1 View Options

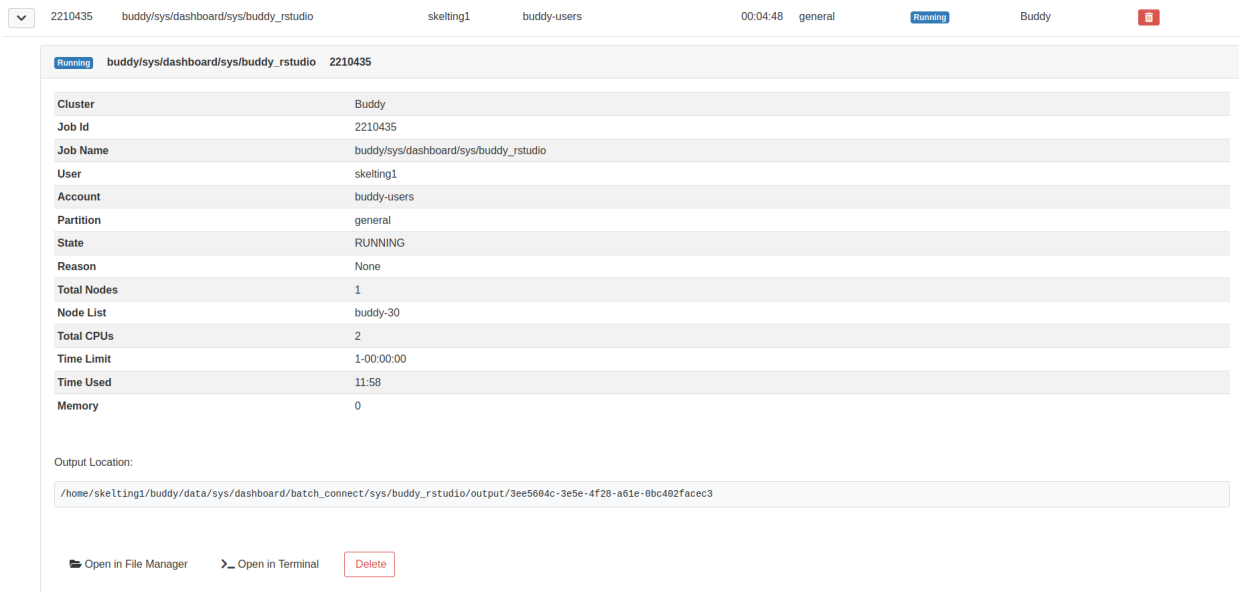
There are a few options to adjust your current view within the Active Jobs application.



- **Your Jobs/All Jobs:** Toggles whether you want to view only your jobs, or all jobs on the cluster
- **All Clusters/Buddy:** Chooses which cluster's active jobs will show. Since there is only one cluster, Buddy, this option is currently irrelevant.
- **Filter:** Filter displayed jobs via keywords
- **Show ## Entries:** Change the number of entries to show on a single page

## 4.6.2 Job Context Menu

All jobs have an associated context menu that can be seen by clicking the arrow next to the job. Note the below screenshot is what will be seen for self-owned jobs. Jobs owned by other users will be lacking many of these options as you don't have permission to modify them.




- **Red Trash Bin:** Cancel your current job

- **Job Status:** Status of your current Slurm job. Possible states include Running, Pending, Completing, and Cancelled. Please see our page on Slurm for more information.
- **Open in File Manager:** Opens the job working directory in the OnDemand file browser. This makes tasks such as viewing job output more convenient.
- **Open in Terminal:** Opens the job working directory in the OnDemand terminal
- **Delete:** Cancels the job selected


## 4.7 Job Composer

The Job Composer can be accessed by going to Jobs>Job Composer and open in a new tab. Job Composer allows for creating and running slurm jobs from within your browser. Some users may find this more convenient than using the terminal to run slurm jobs.

### 4.7.1 Overview


UNIVERSITY OF  
Central Oklahoma

[Job Composer](#)
[Jobs](#)
[Templates](#)

 Help

### Jobs

+ New Job ▾

☆ Create Template

Edit Files

Job Options

Open Terminal

Submit

Stop

Loading

Delete

Show 25 ▾ entries

Search:

Created	Name	ID	Cluster	Status
April 21, 2022 3:01pm	Gaussian Job		Buddy	Not Submitted

Showing 1 to 1 of 1 entries

Previous

1

Next

Job Details

Job Name:

**Gaussian Job**

Submit to:

Buddy

Account:

Not specified

Script location:

/home/skelting1/buddy/data/sys/myjobs/projects/default/1

Script name:

g16\_batch.sh

Folder Contents:

example.com

g16\_batch.sh

Submit Script

g16\_batch.sh

Script contents:

```
#!/bin/bash
#SBATCH --job-name=g16
#SBATCH --output=%j-g16.out
```

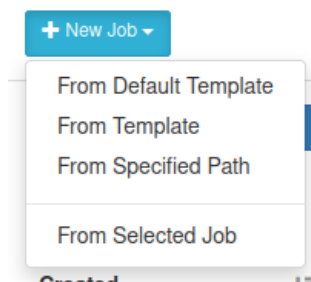
The Job Composer's main screen offers a number of options.

- **+New Job:** Create a new job from one of the following
  - **From Default Template:** Creates a new job from the default template
  - **From Template:** Creates a new job from a pre-defined template
  - **From Specified Path:** Creates a new job from files within a specified directory
  - **From Selected Jobs:** Copies the currently selected job into a new job.
- **Create Template:** Create a new template from the current job. There are a few options that need set.
  - **Path:** Path of folder to be used to generate the template
  - **Name:** Name of the job template
  - **Cluster:** Name of the cluster the job will run on by default. Buddy is the only cluster, so this option is irrelevant.
  - **Notes:** Notes for the job written in HTML markup.
- **Edit Files:** Open the currently selected job in the file browser to edit and manage job files. You can use this to upload new inoput files or modify job scripts
- **Job Options:** Opens a dialogue within your current window to set various job options
  - **Name:** Name of your job
  - **Cluster:** Name of the cluster your job will run on. Since we only have one cluster, Buddy, this option is irrelevant.
  - **Specify Job Script:** Specify the name of the script to be executed when the submit button is pressed within the job composer.
  - **Account:** We do not currently use accounting. Please leave this field blank.
  - **Job Array Specification:** Please see Advanced Slurm topics for more information on configuring arrays.
- **Open Terminal:** Open the current job's folder in the the terminal. This is a faster option for managing job files for those familiar with bash.
- **Submit:** Submit the selected job to the cluster
- **Stop:** Cancel the running of a selected job
- **Delete:** Delete the current job from job composer
- **Job Details:** Details regarding the current job. Use the "Job Options" button to modify these fields.
- **Submit Script:** Details of the script being used. Options in this field include
  - **Open Editor:** Opens the job script in the file editor
  - **Open Terminal:** Opens the job folder in the terminal
  - **Open Dir:** Opens the job folder in the file browser

## 4.7.2 Creating Job From Template

Here we will look at creating a job from a template. The process is roughly the same for a default template as well. This example will use Gaussian. Please see the Software and Slurm sections of the documentation for details on writing Slurm scripts and how to use your desired application.

1. Select “From Template” from the “New Job” menu.



2. From the template screen, select the desired template and click “Create New Job” from the right hand pane. For this example, we will select the Gaussian template. Be sure to also set the desired job name. We will ignore the Cluster option, as Buddy is currently the only available cluster.

View Files

Open Terminal

Delete

Show 10 entries

Search:

Name	Cluster	Source
Default	Buddy	System Templates
Gaussian	Buddy	System Templates
Python Conda	Buddy	System Templates
Quantum Espresso	Buddy	System Templates
Stacks	Buddy	System Templates

Showing 1 to 5 of 5 entries

Previous 1 Next

Create New "Gaussian"

<p>Default Gaussian template. Be sure to upload your input file using the job composer and change the name of your input file in the script.</p>

**Job Name:**

**Cluster:**

Buddy

**Script Name:**

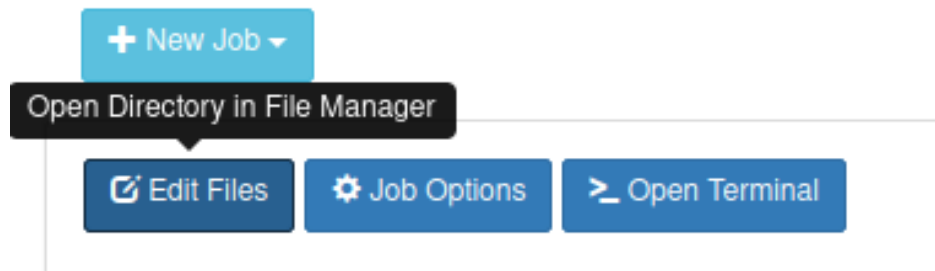
Create New Job

Reset

Selected Template Details

Template location:

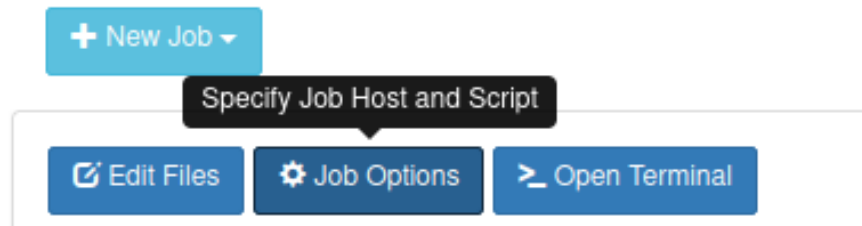
3. We will now need to make some modifications to our script and upload the input file we want. Select your job and click “Edit Files” on the left side. This will open the file browser in a new window. Upload your input files and edit the Slurm script accordingly. Please see the File Browser and File Editor section of the documentation. Please also see the Slurm documentation for writing your script.



4. Once everything is setup, we can click submit job. If everything goes as expected, you should see the job pass through several states, eventually reaching completion. To stop a job, simply press the stop button. Output can be viewed by pressing the “Open Dir” button in the bottom right, or using the edit button in the upper left. Should a job not complete as expected.

### 4.7.3 Modifying Job Options

Certain job options can be modifying by selecting the desired job and clicking “Job Options”.



Job options will be opened within your current window.

## Job Options

## Name

Gaussian Benzene

## Cluster

Buddy

## Specify job script

g16\_batch.sh

Files larger than 65KB are omitted for the job script field

## Account

Account is an optional field. If not set, the account may be auto-set by the submit filter.

## Job array specification

1-10

Job arrays are optional. e.g. 1-10

Save

Reset

Back

- **Name:** Name of your job
- **Cluster:** Name of the cluster your job will run on. Since we only have one cluster, Buddy, this option is irrelevant.
- **Specify Job Script:** Specify the name of the script to be executed when the submit button is pressed within the job composer.
- **Account:** We do not currently use accounting. Please leave this field blank.
- **Job Array Specification:** Please see Advanced Slurm topics for more information on configuring arrays.

You can reset any changes made with the reset button, or by clicking back.

#### 4.7.4 Saving and Managing Templates

User's can save and manage Templates under the "Templates" tab of the Job Composer. System provided Job Composer templates are only available for a limited number of software. We are currently working to add more to OnDemand. Generic Slurm templates are available under the Slurm section and specific templates are available in the SOFTWARE section.



New templates can be created either by starting one from scratch or by copying a current template. Template files can also be modified from this view.



+ New Template

Copy Template

View Files

Open Terminal

Delete

Show 10 entries

Search:

Name	Cluster	Source
Example Template	Buddy	My Templates
Default	Buddy	System Templates
Gaussian	Buddy	System Templates
Python Conda	Buddy	System Templates
Quantum Espresso	Buddy	System Templates
Stacks	Buddy	System Templates

Showing 1 to 6 of 6 entries

Previous 1 Next

Create New "Example Template"

<p>Change these notes by editing the manifest.yml in this template's directory</p>

**Job Name:**

**Cluster:**

**Script Name:**

Create New Job

Reset

Selected Template Details

Template location:

Folder Contents:

**Note:** The scope of this section is **extremely** limited. A more in depth walk through of utilizing the template feature will eventually be provided under the ADVANCED section of this documentation.

- **New Template/Copy Template:** Create or copy a template for use.
  - **Path:** Path that contains your desired default template files. They should include a runtime script, relative input files, and manifest.yml. Please note the manifest.yml, while important, is not covered in this document. Please view another templates folder to see how this is constructed or email UCO's HPC support.
  - **Name:** Name of the template
  - **Cluster:** Name of the cluster your job will run on. Since we only have one cluster, Buddy, this option is irrelevant.
  - **Notes:** Notes about the job script template.
- **Delete:** Delete a selected template. You may only delete user created templates as System Templates are managed by CREIC.



## USING THE TERMINAL

The Linux command line is one of two methods for accessing Buddy resources. It's features, power, and flexibility are essential for those wishing to properly utilize the cluster. While the navigation terminal may seem confusing, many of its aspects are straightforward.

### 5.1 Background

When computer first came about, one of the first operating systems to arrive was Unix. It was designed to run as a multi-user system on mainframe computers, with users connecting to it remotely via individual terminals. These terminals were extremely simplistic and consisted primarily of a keyboard and screen.

Compared to graphics, text is very light on resources. Because of this, older machines could run dozens of terminals even across the slowest of networks. Despite the nature of terminals, users were still able to interact with programs quickly and efficiently. The commands were also kept very short to reduce the number of keystrokes needed, which sped up the use of the terminal even more. This speed and efficiency is one reason why this text interface is still widely used today.

When logged into a Unix mainframe via a terminal users still had to manage the sort of file management tasks that you might now perform with a mouse and a couple of windows. Whether creating files, renaming them, putting them into subdirectories or moving them around on disk, users could do everything entirely with a textual interface.

Buddy utilizes an operating system called Linux. While it bears some similarities to Unix, it is most definitely not identical and has many major differences. Buddy's terminal operates using something called "Bash" for users to communicate via the command line. Bash is widely used on Linux systems, and is well documented. This page will cover some basic functions of Bash, including some simple scripting.

### 5.2 Accessing the Terminal

The terminal can be accessed in one of two ways. One is via the web browser, and the other is via SSH. Upon opening, you will be greeted with a black screen and blinking cursor.

### 5.2.1 Web browser Access

The easiest way to access the terminal is via OnDemand. Once you are logged in, you can access the terminal by going to Clusters>Buddy Shell Access. This will open up a terminal for you in a new window.



You will see the terminal in your web browser once you are logged in

```
Host: buddy.uco.edu
Last login: Thu Apr 21 13:23:47 2022 from buddy.uco.edu
## April 2022 - User Notice

**Maintenance is scheduled for the first and third Thursday of every month. Jobs should continue to run without interruption.**

Jupyter now supports custom conda environments! Simply check advanced and change the version to custom.

VSCode Server has now been added to Buddy! You can access it through your interactive applications. Please note that it is still in beta
, and features such as github are not working. If you notice any issues during use, please email hpc@uco.edu so we can track them.

We are always adding new features and OnDemand applications! If you have any software requests, please let us know at hpc@uco.edu.

---
[skelting1@buddy ~]$
```

### 5.2.2 SSH Access

SSH access is also available for users who desire to use a preferred terminal emulator. You can ssh into buddy by connecting to `username@buddy.uco.edu`. There are few applications available to utilize SSH

---

**Note:** Please read the associated documentation for each of these softwares if you desire to use them. Users uncomfortable with this method of access are recommended to use the built in OnDemand terminal mentioned above for SSH and the OnDemand file browser for uploading and downloading files to Buddy.

---

- Windows

- **Powershell:** Powershell is a built in Windows terminal emulator that uses the Powershell language. You can access it via your start menu and connect Buddy by using the command `ssh username@buddy.uco.edu`. You can exit the ssh prompt by typing `exit`.
- **Putty:** Putty is a popular option for Windows and can be downloaded from the Putty website.
- **MobaXTerm:** MobaXTerm is another common software and can be downloaded from the Moba website.
- **WinSCP:** WinSCP is a software that is not for SSH, but rather file transfer over SCP.

- **OSX**

- **Terminal:** OSX has its own built in terminal emulator. It can be accessed from your utility folder and you can connect to Buddy with the command `ssh username@buddy.uco.edu`. You can exit the ssh prompt by typing `exit`.
- **Finder:** Your file browser in Mac OS can be used to directly connect Buddy for file transfer. You will want to connect to `sftp://username@buddy.uco.edu`.

**Warning:** Filezilla is no longer recommended as its installer comes bundled with other software! While the bundled offer is not malicious, this can be considered undesirable as the bundled application is installed in a deceptive manner and can interfere with your anti-virus.

## 5.3 Terminal Basics

This section will teach everyday commands that will be used regularly in the terminal. On a terminal, you don't have a file browser, word, or any other "GUI" application. But that doesn't mean it is difficult to use. While there is a learning curve, once common commands are memorized, it's as easy as riding a bike.

### 5.3.1 Navigation

Navigating files and folders is a fundamental aspect of using any computer. But within the terminal, we are not automatically shown what we want to see. We have to be more explicit.

Let's start by viewing the contents of our current folder using the "LiSt" command.

```
[skelting1@buddy ~]$ ls
batchjob.sh  Data_Folder_01  Data Folder 02  slurm_output.txt
```

You can view the contents of a directory by passing a file path to the "LiSt"

```
[skelting1@buddy ~]$ ls Data_Folder_01
data-set-01.dat  data-set-02.dat  meta
```

Let's pass a "List" option to our "LiSt" command.

```
[skelting1@buddy ~]$ ls -l
total 4
-rw-r--r-- 1 skelting1 skelting1 635 Apr 21 13:13 batchjob.sh
drwxr-xr-x 2 skelting1 skelting1 10 Apr 21 11:37 Data_Folder_01
drwxr-xr-x 2 skelting1 skelting1 10 Apr 21 11:37 Data Folder 02
-rw-r--r-- 1 skelting1 skelting1 0 Apr 21 11:37 slurm_output.txt
```

There's a lot of information to unpack here. For now, we will share that the date and time shows when a file was modified last.

This is all well and good, but where are we? Let's "Print (our) Working Directory"

```
[skelting1@buddy ~]$ pwd
/home/skelting1/
```

This path is our home folder. The username will of course differ. Your home folder is where all of your files will be stored on Buddy. When you login, this is the first folder you will see. But what if we want to access our other folders? Let's "Change Directory"

```
[skelting1@buddy ~]$ cd Data_Folder_01
[skelting1@buddy Data_Folder_01]$ pwd
/home/skelting1/Data_Folder_01
```

**Warning:** Linux is CaSe SeNsItIvE! Failure to match case will result in your commands not working.

You'll notice our prompt changes to show our current folder. Looking even closer, you'll notice we started with a ~ as our current folder. This is because the ~ is a special symbol to represent our home folder. We can even get back into the home folder by changing our directory to it.

```
[skelting1@buddy Data_Folder_01]$ cd ~
[skelting1@buddy ~]$ pwd
/home/skelting1/
```

Neat! But what if we are several folders in and just want to go up a folder? Let's see how that would work.

```
[skelting1@buddy ~]$ ls
batchjob.sh Data_Folder_01 Data_Folder_02 slurm_output.txt
[skelting1@buddy ~]$ cd Data_Folder_01
[skelting1@buddy Data_Folder_01]$ ls
data-set-01.dat data-set-02.dat meta
[skelting1@buddy Data_Folder_01]$ cd meta
[skelting1@buddy meta]$ ls
info.json
[skelting1@buddy meta]$ pwd
/home/skelting1/~Data_Folder_01/meta
```

One option for going up a folder is to give our cd command the absolute path of the parent directory: /home/skelting1/Data\_Folder\_01/ but, this is highly inefficient. Let's examine another special folder. We will need to add another option to our "LiSt" command to see what they are.

```
[skelting1@buddy meta]$ ls -a
.  ..  info.json
```

The "All" option for ls shows us some directories we couldn't see before. One is a directory named . and the other is a directory named ... . represents the current directory, and .. represents the directory above it. Going up a directory is as easy as

```
[skelting1@buddy meta]$ cd ..
[skelting1@buddy Data_Folder_01]$
```

With the `..` relative path in your tool-belt you can go anywhere by building up a longer path. For example, to jump from a directory to a sibling directory, you could go up a directory and then down with two separate commands or you can jump directly using one command

```
[skelting1@buddy meta]$ pwd
/home/skelting1/Data_Folder_01/meta
[skelting1@buddy meta]$ cd ../../Data_Folder_02
[skelting1@buddy Data_Folder_02]$ pwd
/home/skelting1/Data_Folder_02
```

You'll notice if you try to change directory to `.` that nothing really happens. This is the intended behavior as we are changing directory to our current directory. Which of course leaves us in the same place! Let's go back to our home folder and review a special case you will most likely encounter.

```
[skelting1@buddy ~]$ ls
batchjob.sh Data_Folder_01 Data Folder 02 slurm_output.txt
```

You'll notice that one of our folder names has spaces in it. This is generally not recommended from a convenience standpoint, but it happens often for one reason or another. If we try to `cd` into this folder, odd things happen.

```
[skelting1@buddy ~]$ cd Data Folder 02
-bash: cd: Data: No such file or directory
```

Our `cd` command only wants to take the first argument. In order to read spaces, we have to use what's called an "Escape Character". This is simply a backslash `\`, not to be confused with the forward-slash `/` we use for paths. So how is the escape character used?

```
[skelting1@buddy ~]$ cd Data\ Folder\ 02
[skelting1@buddy Data Folder 02]$
```

This may not seem intuitive to some users, so there is also the option of putting the path in quotes.

```
[skelting1@buddy ~]$ cd "Data Folder 02"
[skelting1@buddy Data Folder 02]$
```

**Note:** You may find yourself annoyed by having to always type out these paths completely. Thankfully, you can use the Tab key to auto-complete. If you press tab and nothing happens, either there is nothing beginning with that name, there are more than one items starting with that particular set of characters, or you've made a syntax error. You may try hitting Tab three times to show available options. Alternatively, backspace over your command and type `ls` and/or `pwd` to ensure you are in the right directory and the item is actually in there.

### 5.3.2 Creating and Deleting Files and Directories

Often, it is needed to make a new directory. To do this we use the "MaKe DIRectory" command. As previously discussed, it is suggested to not name directories with spaces.

```
[skelting1@buddy ~]$ mkdir Data_Folder_03
[skelting1@buddy ~]$ ls
batchjob.sh Data_Folder_01 Data Folder 02 Data_Folder_03 slurm_output.txt
```

To delete a directory, we simply use the "ReMove DIRectory" command.

```
[skelting1@buddy ~]$ rmdir Data_Folder_03
[skelting1@buddy ~]$ ls
batchjob.sh  Data_Folder_01  Data Folder 02  slurm_output.txt
```

This isn't always the best option. Especially considering it fails to work if your directory contains a file within it. For that reason, the "ReMove" command is generally recommended. This works for both files and directories. Notice that to remove a directory, we must pass a "Recursive" option, but a file doesn't require it.

```
[skelting1@buddy ~]$ ls
batchjob.sh  Data_Folder_01  Data Folder 02  Data_Folder_03  slurm_output.txt
[skelting1@buddy ~]$ rm slurm_output.txt
[skelting1@buddy ~]$ ls
batchjob.sh  Data_Folder_01  Data Folder 02  Data_Folder_03
[skelting1@buddy ~]$ rm -r Data_Folder_03
[skelting1@buddy ~]$ ls
batchjob.sh  Data_Folder_01  Data Folder 02
```

If you want to delete several similarly named files, like output files, you can replace the part that differs between the paths by a star which is called a wildcard.

```
[skelting1@buddy ~]$ ls
data-001.out data-002.out data-003.out data-004.out foobar.out example.txt
[skelting1@buddy ~]$ rm data-*.out
[skelting1@buddy ~]$ ls
foobar.out example.out
```

**Warning:** The `rm` command is permanent!! There is no trashcan to restore files from, and data recovery is not possible. Please be careful when using this command. Remember, think twice, hit enter once.

You may notice that if the directory is filled with files, it may prompt you about deleting each and every file. If this is the case, we can use the "Force" option.

```
[skelting1@buddy ~]$ rm -r -f Data_Folder_03
```

or

```
[skelting1@buddy ~]$ rm -rf Data_Folder_03
```

There may be instances when you want to create a blank file. To do this, we use the `touch` command

```
[skelting1@buddy ~]$ touch script.sh
[skelting1@buddy ~]$ ls
batchjob.sh  Data_Folder_01  Data Folder 02  script.sh  slurm_output.txt
```



### 5.3.3 Copy, Move, and Rename

Often times, we want to replicate files and folders on our system. To do this, we use the “CoPy” command. For copying directories that contain files, we also want to include a “Recursive” option. When using copy, we will first specify the source followed by the destination

```
[skelting1@buddy ~]$ cp script.sh copy-of-script.sh
[skelting1@buddy ~]$ ls
batchjob.sh  copy-of-script.sh  Data_Folder_01  Data Folder 02  script.sh  slurm_output.
↳txt
[skelting1@buddy ~]$ cp -r Data_Folder_01 Data_Folder_04
[skelting1@buddy ~]$ ls
batchjob.sh  copy-of-script.sh  Data_Folder_01  Data Folder 02  Data_Folder_04  script.
↳sh  slurm_output.txt
```

We can also copy files or directories into other directories

```
[skelting1@buddy ~]$ cp script.sh Data_Folder_01/script.sh
[skelting1@buddy ~]$ ls Data_Folder_01
data-set-01.dat  data-set-02.dat  meta  script.sh
```

The “MoVe” command is used to move files from one place to another. Its behavior can change depending on what paths you provide and what those paths go to.

If you provide two paths and the first one exist while the second one does not, the first file will be renamed to the second.

```
[skelting1@buddy ~]$ ls
batchjob.sh  copy-of-script.sh  Data_Folder_01  Data Folder 02  script.sh  slurm_output.txt
[skelting1@buddy ~]$ mv script.sh foobar.sh
[skelting1@buddy ~]$ ls
batchjob.sh  copy-of-script.sh  Data_Folder_01  Data Folder 02  foobar.sh  slurm_output.txt
```

**Note:** If foobar.sh already exists, the command will throw an error. Using the -f flag will allow you to overwrite foobar.sh

This works for directories as well with one exception. If the last path provided is an existing directory, whatever is at all of the other paths will be moved into the directory

```
[skelting1@buddy ~]$ ls
batchjob.sh  copy-of-script.sh  Data_Folder_01  Data Folder 02  foobar.sh  slurm_output.txt
[skelting1@buddy ~]$ mv batchjob.sh copy-of-script.sh foobar.sh slurm_output.txt Data_
↳Folder_01
[skelting1@buddy ~]$ ls
Data_Folder_01  Data Folder 02
[skelting1@buddy ~]$ ls Data_Folder_01
data-set-01.dat  data-set-02.dat  meta  script.sh  batchjob.sh  copy-of-script.sh  foobar.
↳sh  slurm_output.txt
```

You can also use wildcards to move similarly named paths as well. If you wanted to move your .dat files back out of Data\_Folder\_1 and into the current working directory you could do the following

```
[skelting1@buddy ~]$ ls Data_Folder_01
data-set-01.dat  data-set-02.dat  meta  script.sh  batchjob.sh  copy-of-script.sh  foobar.
```

(continues on next page)

(continued from previous page)

```
↪ sh slurm_output.txt
[skelting1@buddy ~]$ mv Data_Folder_01/*.dat .
[skelting1@buddy ~]$ ls
data-set-01.dat data-set-01.dat Data_Folder_01 Data Folder 02
```

---

**Note:** Notice the use of the special `.` path to reference the current working directory

---

## 5.4 Common Commands and Features

### 5.4.1 File Viewing/Editing and Pipes

#### Viewing

Viewing the contents of a file is a common task so there are a couple of commands to do so. The “conCATenate” command is used to display the entire contents of a file

```
[skelting1@buddy ~]$ cat data-set-01.dat
key1 value1
key2 value2
key3 value3
```

The name “conCATenate” comes from its ability to join the output of files together. This is evident when viewing the contents of several files at once

```
[skelting1@buddy ~]$ cat data-set-01.dat data-set-02.dat
key1 value1
key2 value2
key3 value3
other data1
other data2
other data3
```

You could have also used the wildcard character here: `cat *.dat`

If you just want to view the beginning or end of a file you can use the `head` and `tail` commands respectively. The `-n` flag specifies how many lines you would like to see; defaulting to 10 if the `-n` flag is omitted

```
[skelting1@buddy ~]$ tail -n 2 data-set-01.dat
key2 value2
key3 value3
[skelting1@buddy ~]$ head -n 2 data-set-01.dat
key1 value1
key2 value2
```

`Tail` has a useful feature in the `-f` flag which causes `tail` to watch for changes in the given files and updates the screen as they occur. This is particularly useful for output files such as those generated by `slurm` scripts

```
[skelting1@buddy ~]$ tail -n 2 -f slurm.out
output line 2
output line 3
```

later...

```
[skelting1@buddy ~]$ tail -n 2 -f slurm.out
output line2
output line3
output line4
```

**Note:** Displaying the contents of a file usually only makes sense since the file is plain text; i.e. not a binary file. Catting out a .bin file will just result in your screen being filled with random nonsense characters

## Editing with nano

To begin editing a file with nano pass its file path to the nano command

```
[skelting1@buddy ~]$ nano data-set-02.dat
```

Listing 1: data-set-02.dat

```
1 April 1st subject baseline
2 May 5th subject reports weight loss
3 June 3rd subject reports light-headedness may be due to trial drug
4 June 10th trial discontinued
```

Navigate using arrow keys and edit the file by typing characters and using backspace like you'd expect. The characters you type will appear before the currently selected character.

**Note:** When using the nano editor, command shortcuts appear at the bottom of the screen. These shortcuts can be confusing if you aren't familiar with the notation so just remember that `^` refers to the `Ctrl` key and `M` represents the meta key also known as the `Alt` key on windows or the `Option` on Mac.

Copy, cut, and paste can be achieved by first selecting the desired text. Move the cursor to the first character of the selection, hold `Shift` and use the arrow keys to select all desired characters. Then press `Alt+6` or `Option+6` to copy or `Ctrl+k` to cut, move to the desired paste location and press `Ctrl+u` to paste.

Save using `Ctrl+o` and exit using `Ctrl+x`.

## Editing with vim

To begin editing a file with nano pass its file path to the vim command

```
[skelting1@buddy ~]$ vim data-set-02.dat
```

Listing 2: data-set-02.dat

```
1 April 1st subject baseline
2 May 5th subject reports weight loss
3 June 3rd subject reports light-headedness may be due to trial drug
4 June 10th trial discontinued
```

Vim uses the concept of “modes” to categorize different kinds of behavior. For instance, typing new portions of a file must happen within “insert mode”. Understanding how to use and switch between modes is one of the most important parts of using vim.

To enter a mode first enter “normal” mode by pressing the ESC key. You can think of this as backing out of whatever other mode you might already be in. Then press the appropriate key to enter your desired mode.

---

**Tip:** If you hit something by mistake and can’t figure out how to get out of it, just spam the ESC key. This will always get you back to vim’s normal mode and you can get where you want from there. When in doubt, get to normal mode.

---

Shortcut	Mode	Description
ESC	Normal mode	manipulate whole lines and jump into other modes
i	Insert mode	insert and delete characters like a normal editor
Ctrl+v	Block visual mode	select character by character and across multiple lines
Shift+v	Linewise visual mode	select multiple lines
:	Command mode	type commands to perform various vim functions

To begin editing a file, make sure you’re in normal mode by pressing ESC, enter insert mode by pressing i, and begin editing.

Cut, copy, and paste can be carried out on entire lines or on selected portions of text.

To copy an entire line, enter normal mode by pressing ESC, navigate the cursor to the desired line, and “yank” the line with yy. Cut is similar to copy but you “delete” the line with dd instead of “yanking” it. To paste the stored line, navigate the cursor to a line near where you would like to “put” it and press p to paste the below the current line or Shift+p to paste above it.

To copy or cut a selection you first need to select some text. To select multiple lines, first be sure you’re in normal mode by typing ESC, then navigate to the first line you would like to select and enter “linewise visual mode” by pressing Shift+v. Finally use the up and down arrow keys to select more lines. To select a block of text, first be sure you’re in normal mode by pressing ESC, then navigate the first character you would like to select and enter “block visual mode” by pressing Ctrl+v. Finally use the arrow keys to complete your selection.

With some of the text selected you can “yank” (copy) the selection with y or “delete”(cut) it with d. You can “put”(paste) the selected text using p or Shift+p. If you made the selection using “linewise visual mode” the selection will be pasted above or below the current line. If you made the selection using “block visual mode” the selection will be pasted before or after the currently selected character.

To save the current file or exit vim you need to use the appropriate vim commands. Firstly, you need to be sure you are in normal mode by typing ESC then jump into command mode by typing :. In command mode you can save the current file by typing the w command and hitting enter. To save and quit type wq and hit enter. You can quit without saving by just using the q command if you haven’t edited the file otherwise you will have to force vim to exit without saving by using the q! command.

## Pipes

Sometimes you may want to take the output of a command and do something with it like storing it in a file or passing it to another command; this is where pipes become useful

For example: if you wanted to store the result of the ls command as a file, you could do the following

```
[skelting1@buddy ~]$ ls > output.txt
[skelting1@buddy ~]$ cat output.txt
data-set-01.dat data-set-02.dat meta
```

The `>` pipe places the output of the previous command into the given file; creating it if it doesn't already exist and overwriting it if it does. If you wish to append the command output to the end of the given file instead of overwriting it use the `>>` pipe.

```
[skelting1@buddy ~]$ ls >> output.txt
[skelting1@buddy ~]$ cat output.txt
data-set-01.dat data-set-02.dat meta
[skelting1@buddy ~]$ ls >> output.txt
[skelting1@buddy ~]$ cat output.txt
data-set-01.dat data-set-02.dat meta
data-set-01.dat data-set-02.dat meta
```

To pass the output of one command as input to another use the `|` pipe. For example if you only wanted the last two lines of the `ls -a` command you could do the following.

```
[skelting1@buddy ~]$ ls -l
total 16
drwxr-xr-x 2 tdunn3 tdunn3 4096 May  4 11:35 Data_Folder_01
drwxr-xr-x 2 tdunn3 tdunn3 4096 May  4 11:35 'Data Folder 02'
-rw-r--r-- 1 tdunn3 tdunn3   36 May  4 11:34 data-set-01.dat
-rw-r--r-- 1 tdunn3 tdunn3   39 May  4 11:34 data-set-02.dat
[skelting1@buddy ~]$ ls -l | tail -n 2
-rw-r--r-- 1 tdunn3 tdunn3   36 May  4 11:34 data-set-01.dat
-rw-r--r-- 1 tdunn3 tdunn3   39 May  4 11:34 data-set-02.dat
```

It is common practice to chain multiple commands one after the other using `|` in order to refine data with successive commands.

## 5.4.2 Shortcuts

Command	description
!!	used within a command will be replaced with your last run command
Ctl+Shift+c	copies the selected text in the terminal
Ctl+Shift+v	pastes in terminal
Tab	completes the portion of text that has already been typed
Up	autofills the last command
Home	jumps to beginning of line
End	jumps to end of line

## 5.4.3 Searching

### grep

The `grep` command is one of the most useful commands you could have in your arsenal. It's used to search for words or patterns within one or multiple files or strings.

To search for lines containing a word or phrase within a file:

```
[skelting1@buddy ~]$ cat data-set-02.dat
April 1st subject baseline
May 5th subject reports weight loss
```

(continues on next page)

(continued from previous page)

```

June 3rd subject reports light-headedness may be due to trial drug
June 10th trial discontinued
[skelting1@buddy ~]$ grep data-set-02.dat May
May 5th subject reports weight loss

```

Grep also works with data piped in from other commands. This is actually a common use case for grep

```

[skelting1@buddy ~]$ ls -l
total 16
drwxr-xr-x 2 tdunn3 tdunn3 4096 May  4 11:35 Data_Folder_01
drwxr-xr-x 2 tdunn3 tdunn3 4096 May  4 11:35 'Data Folder 02'
-rw-r--r-- 1 tdunn3 tdunn3   36 May  4 11:34 data-set-01.dat
-rw-r--r-- 1 tdunn3 tdunn3   39 May  4 11:34 data-set-02.dat
[skelting1@buddy ~]$ ls -l | grep 02
drwxr-xr-x 2 tdunn3 tdunn3 4096 May  4 11:35 'Data Folder 02'
-rw-r--r-- 1 tdunn3 tdunn3   39 May  4 11:34 data-set-02.dat

```

Another important use case for grep is listing files that contain a pattern with the `-l` flag. It is often used with the `-i` flag which makes the search case insensitive (May is the same as may) and `-R` which recursively searches directories.

```

[skelting1@buddy ~]$ grep -Ril may
data-set-02.dat

```

**Note:** To search for patterns that contain spaces you can either “escape” the spaces using back slashes, i.e. *my string* becomes *my\ string*, or you can quote them, *my string* becomes “*my string*”. However, quotes can be tricky because they are designed to search for regular expressions so special characters like `.`, `'`, `s` and `-` are treated as instructions. To search for quoted patterns containing special characters you must “escape” them with back slashes like “my hyper\strange expression goes \[HERE\] \.” This forces those characters to be read as “literals” meaning they are treated as the literal character you typed and not a regular expression instruction.

## find

The find command is used to find files matching the given pattern

To find files within the current directory and its subdirectories with “02” in the name

```

[skelting1@buddy ~]$ ls
Data_Folder_01 'Data Folder 02' data-set-01.dat data-set-02.dat
[skelting1@buddy ~]$ find . -name 02
'Data Folder 02' data-set-02.dat

```

Find is often used with other commands. For example you can find with the type `f` (regular file) using find’s `-type` and then of pass those to grep using the `-exec` flag to select only those files that contain a certain phrase

```

[skelting1@buddy ~]$ find . -type f -exec grep -l May {} \;
data-set-02.dat

```

The `{}` in the above command will be filled by the output of the rest of the find command. So actual command that returns the above output is `grep -l May data-set-01.dat data-set-02.dat`

## 5.4.4 File Permissions and Information

### Listing file info

The `ls -l` command provides a lot of useful information.

```
[skelting1@buddy ~]$ ls -l
total 16
drwxr-xr-x 2 tdunn3 tdunn3 4096 May  4 11:35 Data_Folder_01
drwxr-xr-x 2 tdunn3 tdunn3 4096 May  4 11:35 'Data Folder 02'
-rw-r--r-- 1 tdunn3 tdunn3   36 May  4 11:34 data-set-01.dat
-rw-r--r-- 1 tdunn3 tdunn3   39 May  4 11:34 data-set-02.dat
```

The first part of each line, `drwxr-xr-x`, describe the file permissions i.e. who is allowed to do what with this file. The `d` means we are looking at a directory and the next nine character correspond to the read(`r`), write(`w`), and execute(`x`) permissions for the owner the file, the access group, and everyone else respectively. For example, the first file has permissions of `rw-r--r--` which means the owner can read, write and execute(`rw`) and the group along with everyone else can only read and execute(`r-x`).

The second part counts the number of hard links to the file. If you're curious you can find more information [here](#)

The third and fourth parts correspond to the owner and the group respectively. These are the same owner and group that are referenced in the first part.

The fourth, fifth, and sixth parts describe the file size, the date the file was created, and its name respectively.

### Changing file permissions

The file permissions described in the previous section can be altered with the `chmod` command. There are two ways to use this command: the first is easier to understand but bulky while the second is more obscure but also more concise.

The bulky version:

```
[skelting1@buddy ~]$ chmod u=rwx,g=rx,o=r data-set-02.dat
```

In the above example `u=rwx` gives read, write, and execute permissions to the owner of the file, `g=rx` gives read and execute permissions to the file's group and `o=r` gives read permissions to everyone else.

For the concise version each of the permissions are encoded as powers of two. Read (`r`) is encoded as 4, write(`w`) is 2, execute(`x`) is 1, and 0 is reserved for no permissions. In the above example, the owner of the file has permissions of `rw`. To express that using the encoded numbers, simply add them together.  $4+2=6$  so `rw` is equivalent to 6. Next, the group has `rx` permissions so,  $4+2=6$ , `rx` is equivalent to 6. Finally everyone else only has read permissions so that is simply 4. The final permissions are written in owner-group-other order so the permissions for this file are written as `764`,

```
[skelting1@buddy ~]$ chmod 764 data-set-02.dat
```

This method may seem complicated but it is a more direct way of representing permissions and it is much more common than the first method so you are more likely to see it when searching for command help than the first example. As with any command, if you use it often enough you will learn it, otherwise don't be ashamed to look it up.

### 5.4.5 Compressing and Uncompressing

Compressing a file makes it more portable; both because it can reduce the file size but also because it bundles a directory of files into a single file.

Probably the easiest way to compress a file is to use the `zip` command.

```
[skelting1@buddy ~]$ zip zipped_file.zip my_directory/
```

To uncompress a zip file use the `unzip` command.

```
[skelting1@buddy ~]$ unzip zipped_file.zip
```

Another popular method for compressing files is the `tar` command. The `tar` command has notoriously difficult to remember flags which has lead to an inside joke among linux users: “Oh so you say you’re a linux expert, but can you tar a file without looking it up?”. This is mainly because no one uses the command frequently enough to remember how it works. That being said, it isn’t difficult; just hard to remember so don’t be afraid to refer back to this guide if you forget because even the experts have to look it up.

```
[skelting1@buddy ~]$ tar -czvf compressed.tar.gz my_directory/
```

Technically, `tar` isn’t a compression tool. It creates archives, called tarballs, which could be compressed but don’t have to be. These archives essentially bundle different files together into a single file. In the above command the `-c` flag creates an archive, `-z` compresses it using the `gzip` format, `-v` displays the progress in the terminal, and `-f` allows you to specify the file name of the final tarball.

Untarring a file can be accomplished as follows:

```
[skelting1@buddy ~]$ tar -xvf compressed.tar.gz
```

In the above example: `-x` extracts the files from the tarball, `-v` displays progress in the terminal, and `-f` allows you to specify the input file name.

### 5.4.6 Downloading Files

Downloading files from sources on the internet is a crucial part of modern terminal usage and there are several ways to accomplish it for different use cases.

---

**Note:** Please review the [Cluster Usage: Rules and Guidelines](#) section and the [Data Transfer](#) section before moving data onto buddy

---

#### wget

Wget is a command line tool for pulling files from a download link. To use it, just navigate to the directory that you wish the file to be downloaded to, copy the link to your file, and use the following command.

```
[skelting1@buddy ~]$ cd my_directory
[skelting1@buddy my_directory]$ wget https://www.google.com/images/branding/googlelogo/
↪2x/googlelogo_light_color_272x92dp.png
```

This example downloads the google banner image.



**Warning:** Since wget allows you to pull data from another source, it can be dangerous. It is important to be sure your download link is correct, you know what you are downloading and you verify the correct file has been downloaded

## sftp

Sftp is a file transfer protocol with a utility similar to ssh which we discussed in a previous section.

```
[myuser@mycomputer ~]$ sftp buddyUsername@buddy.uco.edu
sftp>
```

**Note:** You are able to use the `ls` and the `cd` commands within the sftp prompt so basic navigation is possible.

To move files from your end to buddy first navigate to the directory where your file is and then move the file to buddy using sftp and the `put` command.

```
[myuser@mycomputer ~]$ ls
myfile1.txt myfile2.dat foobar/
[myuser@mycomputer ~]$ sftp buddyUsername@buddy.uco.edu
sftp> put myfile1.txt
Uploading myfile1.txt to /home/buddyUsername/myfile.txt
myfile.txt          100% 236KB  4.5MB/s   00:00
```

Files can be retrieved from a remote server by connecting to it via sftp and using the `get` command. The file will be downloaded to whichever directory you were in when you connected via sftp.

```
sftp> get myfile1.txt
Fetching /home/buddyUsername/myfile1.txt to myfile1.txt
myfile1.txt          100% 236KB 34.4KB/s   00:06
```

To leave an sftp session use the `exit` command.

```
sftp> exit
[myuser@mycomputer ~]$
```

## git

See github section

## 5.5 Tips and Tricks

Some rapid fire tips and tricks

1. If you use a command often enough, you will learn it. Otherwise don't be ashamed to look it up. There is no point in memorizing something you'll never use and even experienced linux users look things up regularly so you're in good company
2. Tab complete is your friend. Just type enough of something to uniquely identify it and then press tab to fill in the rest. It saves so much time

3. The Home key allows you to jump to the beginning of a line while the End key jumps to the end.
4. If you want to stop a command from running before it finishes or you want to get out of something try pressing `Ctrl+c`. This sends a keyboard interrupt which should tell the command to halt
5. When you are first learning the terminal and using different commands it is common to get stuck inside of something and not know how to get out of it. When this happens it is common practice to spam the keys that are most commonly used to escape various programs. Try `q` for quit, `ESC` for escape, `Ctrl+c` sends a keyboard interrupt and should kill the command you are inside of, `Ctrl+d` sometimes works if `Ctrl+c` fails, if you are able to type try typing “exit” or “quit”, and if all else fails just close the terminal window and open a new one; we’ve all had to do that once or twice

## 5.6 Basic Bash Scripting

Sometimes you may find that you need to run several commands one after another or even with some additional logic like branching or loops; this is where scripting becomes useful. The idea is to write down a series of commands within a logical structure in a file and then execute the file just like a normal program. Scripting makes it possible to handle complex scenarios in a repeatable way which is why we use them to submit jobs to the cluster using slurm. Though bash scripts can be executed directly, scripts on Buddy must be run using slurm. See [the slurm section](#) for more information.

### 5.6.1 Hello World

Every script begins with a shebang, `#!/`, followed by the path to the appropriate interpreter which is `/bin/bash` in this case so our hello world script will begin like so:

Listing 3: hello\_world.sh

```
1 #!/bin/bash
```

To print “hello world” to the terminal we can use the `echo` command. Lines beginning with a hash, `#`, are comments. They are not executed by the interpreter and are just for the benefit of anyone reading the code. We will add a comment to label our simple script.

Listing 4: hello\_world.sh

```
1 #!/bin/bash
2
3 # This statement prints hello world to the terminal
4 echo "hello world"
```

### 5.6.2 Basic Logic

**Warning:** Spaces and newlines are very important parts of the bash syntax. Something as simple as adding a space or forgetting to add one can cause a difficult to find error so pay attention to leading and trailing spaces in the following examples

#### Variables

An essential part of any programming language is how variables are handled.

Listing 5: variables.sh

```

1  #!/bin/bash
2
3  # Declare variable in bash. Notice: No space before or after the =
4  my_variable=8
5
6  #reference variable in bash
7  echo $my_variable

```

It is important to note that variables in bash are untyped. You can treat them as strings that are interpreted depending on the situation

## Branching and Conditions

One of the most ubiquitous and most useful programming structures are branching statements which decide which code block to run based on the provided condition.

Here are some of the conditions available in bash

condition	description
\$a -eq \$b	returns true if a and b are equal (both are numbers)
\$a -lt \$b	returns true if a is less than b (both are numbers)
\$a -gt \$b	returns true if a is greater than b (both are numbers)
\$a == \$b	returns true if a and b are equivalent (both are strings)
\$a != \$b	returns true if a and b are not equivalent (both are strings)
! [ condition ]	returns true if condition is false
-d \$a	return true if directory at path a exists
-e \$a	return true if file at path a exists
-r \$a	return true if file at path a exists and can be read
-w \$a	return true if file at path a exists and can be written to
-x \$a	return true if file at path a exists and can be executed
-z "\$a"	return true if variable a is defined

Bash branches might look a little strange if you have used another programming language like python or java.

Listing 6: branching.sh

```

1  #!/bin/bash
2
3  # branching. Note the spaces before and after the condition
4  if [ condition ]
5  then
6      echo condition is true
7  elif [ condition2 ]
8  then
9      echo condition is false and condition2 is true
10 else
11     echo condition and condition2 are false
12 fi

```

There are a few things to unpack here. Firstly the if block ends with `fi`. Statement blocks in bash end with the block name spelled backwards. Secondly after a conditional statement like `if` and `elif` the body is declared with a `then`

statement. Finally, statements are separated by new lines. However, multiple statements can be declared on the same line by separating them with a `;`. This feature allow us to rewrite the above example in a different way which you do tend to see if you search for examples online. The style you choose is up to you but here is how the above example looks making use of `;`.

Listing 7: branching.sh

```
1  #!/bin/bash
2
3  # branching
4  if [ condition ]; then
5      echo condition is true
6  elif [ condition2 ]; then
7      echo condition is false and condition2 is true
8  else
9      echo condition and condition2 are false
10 fi
```

---

**Note:** conditions in bash can get complicated when you start to branch out to using different “test constructs” like `(( ))` and `[]` and unusual operators like `-z` which checks if a variable is defined. Don’t worry about learning these until you run into a situation that requires them

---

## Loops

Loops are essential for any programming language and bash has three varieties: while loops, until loops, and for loops. All loop blocks begin with `do` and end with `done`.

Listing 8: loops.sh

```
1  #!/bin/bash
2
3  ### The following loops are equivalent
4
5  # While loop
6  a=0
7  while [ $a -le 5 ]; do
8      echo a equals: $a
9      ((a=a+1))
10 done
11
12 # until loop
13 a=0
14 until [ $a -ge 5 ]; do
15     echo a equals: $a
16     ((a=a+1))
17 done
18
19 # For loop
20 # Note that the variable "a" after the "for" does not come after a "$".
21 # This is because a is being declared here and is set equal to each
22 # value in the sequence "0 .. 4" one after the other.
```

(continues on next page)

(continued from previous page)

```

23 for a in {0 .. 4} ; do
24     echo a equals: $a
25 done

```

**Note:** in the for loop example, `..` declare a range from 0 up to and including 4. If it is equivalent to just typing `0 1 2 3 4`.

### 5.6.3 Providing Input

Passing input to a script from the command line is as simple as including your input, separated by spaces, after you invoke the script.

```
[skelting1@buddy ~]$ sbatch input_example.sh input1 input2 input3 input4
```

Then each input will be available within your script through number variables along with the name of the script in the `0` number variable.

Listing 9: input\_example.sh

```

1  #!/bin/bash
2
3  echo this script is called: $0
4  echo input one is: $1
5  echo input two is: $2
6  echo input three is: $3
7  echo input four is: $4

```

All inputs and the name of the script can also be accessed in an array `$@`.

Listing 10: input\_example.sh

```

1  #!/bin/bash
2
3  echo input command: $@

```

### 5.6.4 Troubleshooting

**Basic debugging involves:**

1. Identifying the error message
2. Determining the source of the bug
3. Fixing the bug

Identifying the error message may or may not be easy depending on how clear the output of your command is and how much output there is. There is a tendency for error messages to get buried in the output of software if it is poorly written or simply complex. Take time and read carefully to figure out what the computer is trying to tell you about what has gone wrong.

---

**Note:** Warnings are different from errors in that they indicate bad practice, outdated commands, or other minor issues while errors indicate serious issues. They can sometimes hint at the source of an error but that is after you have looked into any error messages that you find.

---

Determining the source of the bug usually involves some knowledge of how your program is constructed though some error messages will tell you the line number that the error occurred on (and we are incredibly grateful for that). A powerful technique is to simply copy the error message (or part of the message) into a web browser and search for other people who have had the same problem. This is usually the first step if you're someone else's software and is very common even and especially with professional programmers. Internet searching is a skill in and of itself so we won't go into detail except to say that there is plenty of helpful information out there as well plenty of misleading information. Be careful when following someone's advice and even then, take precautions, make backups and contact administration if you are unsure or have any questions.

Fixing the bug is the hard part. If you misidentified the actual error message or the real cause of the error then you will make a fix and it won't work or might even break something else. Part of working through this last step is going back to the previous two and trying something else. Reread the error message, try changing the way you phrase your search, gather more information, and try again. Debugging can often feel like you're battling with the computer and it is refusing to yield. If you get stuck or have any questions contact administration with your bug and some of the information you collected and we will try to help you narrow down your issue.

## 6.1 Terminology

### 6.1.1 Tasks

A task is a command which is run in parallel by slurm using srun. Tasks can be used to run more than one command at the same time instead of one after the other, increasing performance

### 6.1.2 Partitions

Partitions are an organizational structure in slurm which allows nodes to be grouped together and for certain options and restrictions to be placed on them. We have a few partitions on Buddy:

---

**Tip:** this information can be found using the `sinfo` command in the terminal

---

Name	Time limit	Description
general	5 days	Used for most jobs
general-long	30 days	Used for jobs that are expected to run for a long time
high-mem	5 days	Used for jobs which are expected to have high memory usage
high-mem-long	30 days	Used for long jobs which are expected to have high memory usage
gpu	5 days	Used for gpu jobs
gpu-long	30 days	Used for gpu jobs which are expected to run for a long time
testing	2 days	Reserved for our internal testing

We recommend that you use the partition that is most appropriate to your application.

### 6.1.3 Cores

Each node has 20 cores so the product of `--tasks-per-node` and `--cpus-per-task` should not exceed 20

## 6.2 Commands

sbatch used allocate resource and run the given script using slurm srun used withing an sbatch file to run a command as a parallel task smap displays the jobs currently running on the cluster sinfo displays information about down and running nodes aswell as partition information

## 6.3 Sbatch Parameters

Sbatch parameters are used to control the way jobs are submitted and run on buddy

### 6.3.1 Common sbatch parameters

Name	Environment variables	Default	Description
-J,--job-name	SLURM_JOB_NAME	Script name or “sbatch”	the name of your job
-o,--output	N/A	“slurm-%j.out”	file to dump standard output of program
-e,--error	N/A	“slurm-%j.out”	file to dump standard error of program
-n,--ntasks	N/A	1 unless --cpus-per task is set	the maximum number of tasks sbatch should allocate resources for
-N,--nodes	SLURM_JOB_NUM_NODES	Minimum nodes to satisfy the -n and -c options	the number of nodes to allocate. A minimum and maximum can also be set like: --nodes=10-12
-c,--cpus-per-task	SLURM_CPUS_PER_TASK	Number of processor per task	the number of cpus to allocate for each task
--ntasks-per-node	SLURM_TASKS_PER_NODE		the number of tasks to allocate for on each node
-p,--partition	SBATCH_PARTITION	General	the partition to run the job in
-t,--time	SBATCH_TIMELIMIT	Max time for partition	the maximum amount of time the job is allowed to run



## COMMON TOOLCHAINS

### 7.1 Component versions in foss toolchain

<i>foss</i>	<i>date</i>	<i>binu-tils</i>	<i>GCC</i>	<i>Open MPI</i>	<i>Flexi-BLAS</i>	<i>Open-BLAS</i>	<i>LAPACK</i>	<i>ScaLA-PACK</i>	<i>FFTW</i>
2019a	Jan '19	2.31.1	8.2.0	3.1.3	(none)	0.3.5	(incl. with Open-BLAS)	2.0.2	3.3.8
2019b	Sept '19	2.32	8.3.0	3.1.4	(none)	0.3.7	(incl. with Open-BLAS)	2.0.2	3.3.8
2020a	May '20	2.34	9.3.0	4.0.3	(none)	0.3.9	(incl. with Open-BLAS)	2.1.0	3.3.8
2020b	Nov '20	2.35	10.2.0	4.0.5	(none)	0.3.12	(incl. with Open-BLAS)	2.1.0	3.3.8
2021a	May '21	2.36.1	10.3.0	4.1.1	3.0.4	0.3.15	(incl. with Open-BLAS)	2.1.0	3.3.9
2021b	Oct '21	2.37	11.2.0	4.1.1	3.0.4	0.3.18	(incl. with Open-BLAS)	2.1.0	3.3.10
2022a	Jun '22	2.38	11.3.0	4.1.4	3.2.0	0.3.20	(incl. with Open-BLAS)	2.2.0	3.3.10
2022b	Dec '22	2.39	12.2.0	4.1.4	3.2.1	0.3.21	(incl. with Open-BLAS)	2.2.0	3.3.10

### 7.2 Component versions in intel toolchain

<i>intel</i>	<i>date</i>	<i>binutils</i>	<i>GCC</i>	<i>Intel compilers</i>	<i>Intel MPI</i>	<i>Intel MKL</i>
2019a	Jan '19	2.31.1	8.2.0	2019.1.144	2018.4.274	2019.1.144
2019b	Sept '19	2.32	8.3.0	2019.5.281	2018.5.288	2019.5.281
2020a	May '20	2.34	9.3.0	2020.1.217	2019.7.217	2020.1.217
2020b	Nov '20	2.35	10.2.0	2020.4.304	2019.9.304	2020.4.304
2021a	May '21	2.36.1	10.3.0	2021.2.0	2021.2.0	2021.2.0
2021b	Oct '21	2.37	11.2.0	2021.4.0	2021.4.0	2021.4.0
2022a	Jun '22	2.38	11.3.0	2022.1.0	2021.6.0	2022.1.0
2022b	Dec '22	2.39	12.2.0	2022.2.1	2021.7.1	2022.2.1

## 7.3 Component versions in foss toolchain (deprecated versions)

<i>foss</i>	<i>date</i>	<i>binu- tils</i>	<i>GCC</i>	<i>Open MPI</i>	<i>Flexi- BLAS</i>	<i>Open- BLAS</i>	<i>LAPACK</i>	<i>ScaLA- PACK</i>	<i>FFTW</i>
2014b	Jul '14	'(none)	4.8.3	1.8.1	(none)	0.2.9	3.5.0	2.0.2	3.3.4
2015a	Jan '15	'(none)	4.9.2	1.8.4	(none)	0.2.13	3.5.0	2.0.2	3.3.4
2015b	Jul '15	2.25	4.9.3	1.8.8	(none)	0.2.14	3.5.0	2.0.2	3.3.4
2016a	Jan '16	2.25	4.9.3	1.10.2	(none)	0.2.15	3.6.0	2.0.2	3.3.4
2016b	Jul '16	2.26	5.4.0	1.10.3	(none)	0.2.18	3.6.1	2.0.2	3.3.4
2017a	Jan '17	2.27	6.3.0	2.0.2	(none)	0.2.19	3.7.0	2.0.2	3.3.6(- pl2)
2017b	Jul '17	2.28	6.4.0	2.1.1	(none)	0.2.20*	(incl. with Open- BLAS)	2.0.2	3.3.6(- pl2)
2018a	Jan '18	2.28	6.4.0	2.1.2	(none)	0.2.20*	(incl. with Open- BLAS)	2.0.2	3.3.7
2018b	Jul '18	2.30	7.3.0	3.1.1	(none)	0.3.1	(incl. with Open- BLAS)	2.0.2	3.3.8

## 7.4 Component versions in intel toolchain (deprecated versions)

<i>intel</i>	<i>date</i>	<i>binutils</i>	<i>GCC</i>	<i>Intel compilers</i>	<i>Intel MPI</i>	<i>Intel MKL</i>
2014b	Jul '14	'(none)	4.8.3	2013.5.192	4.1.3.049	11.1.2.144
2015a	Jan '15	'(none)	4.9.2	2015.1.133	5.0.2.044	11.2.1.133
2015b	Jul '15	2.25	4.9.3	2015.3.187	5.0.3.048	11.2.3.187
2016a	Jan '16	2.26	4.9.3	2016.1.150	5.1.2.150	11.3.1.150
2016b	Jul '16	2.26	5.4.0	2016.3.210	5.1.3.181	11.3.3.210
2017a	Jan '17	2.27	6.3.0	2017.1.132	2017.1.132	2017.1.132
2017b	Jul '17	2.28	6.4.0	2017.4.196	2017.3.196	2017.3.196
2018a	Jan '18	2.28	6.4.0	2018.1.163	2018.1.163	2018.1.163
2018b	Jul '18	2.30	7.3.0	2018.3.222	2018.3.222	2018.3.222

## DATA STORAGE

### 8.1 Home Folder

Every user is given 500Gb of storage in their home directory. If more storage is required consider using scratch storage or contact administration to see about allocating more space.

### 8.2 Project Space

If you are working with a group of users you might want to consider using a project space. Upon request we will create a directory and grant a group of users access to it with one user being the owner. We will also adjust read and write permissions to suit the needs of your group.

### 8.3 Scratch Storage

If you have a large amount of data that needs to be stored or if you have data that is generated and doesn't need to be stored for a long time consider using scratch storage. Scratch storage is large but volatile to data stored here needs to be backed up off site or disposable.



## **DATA TRANSFER**

Data transfer to and from buddy can be done in several ways depending on your needs and the size of the data.

### **9.1 Using SFTP or SCP**

Small files can be transferred to and from Buddy using a file transfer protocol such as SFTP or SCP however, for files greater than 2Gb, please use Globus to increase upload speed and to avoid slowing down the cluster for other users.

### **9.2 Using Github**

For source code it might be preferable to use git to keep data on Buddy synced with a project. This is accomplished by committing changes on your local machine, pushing them up to a repository, and pulling the changes down onto Buddy. However, if the file size is greater than 2Gb, please use Globus to increase upload speed and to avoid slowing down the cluster for other users.

### **9.3 Using Globus**

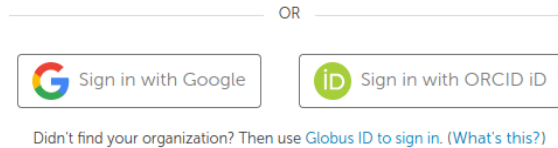
For files that are larger than 2Gb, Globus is the preferred method for moving data between the cluster and another source. This can be accomplished by doing the following:

#### **9.3.1 1. Make an Account and Obtain a Globus ID**

1. Visit [globusid.org](https://globusid.org)
2. Click create a Globus ID
3. Fill out account details and for organization be sure to enter “University of Central Oklahoma”

### 9.3.2 2. Log in with Globus ID

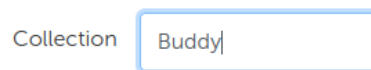
1. Visit [globus.org](https://globus.org) and click Log In in the upper right
2. Click Globus ID to sign in



3. Sign in using your globus username and password. (You may not need sign in if you just created your account and a login cookie still exists in your browser)
4. If this is your first time logging in:
  - a. Enter a verification code from your globus verification email
  - b. Click Continue
  - c. Click Continue again
5. Click Allow

### 9.3.3 3. Connect to Buddy

1. Search Buddy in the collection search bar



2. Select Home storage for files smaller than 100Gb otherwise select DTN storage
3. Click Continue
4. Choose to Link an identity from Buddy OIDC SERVER...
5. Sign in using your Buddy username and password
6. Click Allow

You should now be able to access your buddy home folder or DTN storage. The steps to access both are the same.

## ACCOUNT AND SOFTWARE REQUESTS

### 10.1 General Account Request

A general account is an account made for professors or researchers and their collaborators and is the most given normal permissions on buddy. To make a general account request please contact [administration](#) and provide us with the following information:

- Requester's Name
- Requester's UCO Username (if you are from another university please provide the username provided by your institution and its name)
- Requester's Department
- Required software
- **Do you need a project space?**
  - Are there any special permissions required for this directory for example: "I don't want collaborators to make files", "I want collaborators to be able to read and make files", etc.
- **Professor or student**
  - **If professor: Do you have any students you need to add?**
    - \* Username
    - \* Possibly email
- If you are collaborating with UCO faculty please provide the names and email addresses of your collaborators

### 10.2 Classroom Account Request

A classroom account is one given to students if they will be using Buddy as part of a class. The instructor will be given a general account while the students will receive classroom accounts which expire after the class ends and which have more limited functionality than general accounts. To make a classroom account request please contact [administration](#) and provide us with the following information:

- Class CRN
- Class name
- A two column spreadsheet with student name and student email
- **Do you need a shared classroom space?**

- Are there any special permissions required for this directory for example: “I don’t want students to make files”, “I want students to be able to read and make files”, etc.

## 10.3 Software Request

To make a software request please contact [administration](#) with the name of the software and version information or other requirements if they exist.

---

**Note:** While we try to provide requested software this might not always be possible or may take a long time to work through installation and integration issues. We will keep you updated as much as possible throughout the process and will try to work with you to fulfill your needs. That being said, most software can be installed quickly and easily.

---



## TIPS AND TRICKS

### 11.1 Slurm

#### 11.1.1 Use %j-PROJECT.out for output files

When a slurm job is submitted, the job number is printed to the screen. With this naming scheme it is easy to see which outfile is which and if you're using a terminal you can copy the job number, paste it in the terminal window, and press tab to autofill the rest of the file name.

#### 11.1.2 Create general scripts

If you often run the same command with different input files as is common with python, create a general slurm script that takes in a file path and passes it to that command. For example:

Listing 1: PROJECT\_DIR/scripts/job.sh

```
#!/bin/bash
#SBATCH --job-name=autoencoder
#SBATCH --nodes=1
#SBATCH --cpus-per-task=20
#SBATCH --output=script_outputs/%j-autoencoder.out
#SBATCH --partition=general

# script from commandline
script=$1

# load required packages
module purge
module load TensorFlow/2.6.0-foss-2021a
module load scikit-learn/0.24.2-foss-2021a
module load matplotlib/3.4.2-foss-2021a

# run script
python3 -u $script
echo Finished
```

Then run `sbatch scripts/job.sh PYTHON_SCRIPT` from your project directory. This method can also be used to pass arguments to commands as anything after `sbatch job.sh` is passed to the script.

## 11.2 Python

### 11.2.1 Use the `-u` flag

By default python buffers its outputs. If you use `print()` for debugging, it may be helpful to have output displayed immediately instead of being buffered

---

## CHAPTER TWELVE

---

### OVERVIEW

---

**Note:** This section is incomplete. New software pages will be added overtime, however if you encounter any issues in the mean time, please contact [administration](#).

---



---

CHAPTER  
**THIRTEEN**

---

**ANSYS**



---

CHAPTER  
**FOURTEEN**

---

**COMSOL**





## ANACONDA

### 15.1 Example Script

```
#!/bin/bash
#SBATCH --job-name=my-conda-job
#SBATCH --output=%j-my-conda-job.out
#SBATCH --nodes=1

### USAGE:
# 1. Replace your_environment and your_file
# 2. Submit job

### Load modules
module purge # unload all modules
module load Anaconda3/2020.11 # load anaconda

### Setup
# setup and activate conda environment
source ~/.bashrc # load .bashrc file to setup conda path
conda activate your_environment # activate environment

### Run
# The -u option forces python output to be unbuffered. Useful for debugging.
python3 -u your_file
```



## 16.1 Example Script

```
#!/bin/bash
#SBATCH --job-name=g16
#SBATCH --nodes=2
#SBATCH --cpus-per-task=20
#SBATCH --output=g16-%j.out
#SBATCH --partition=general

### Of the batch options, it is only recommended to change "--job-name", "--nodes", and
### "--output". Any other modifications may result in an error.

### It is only recommended to change the input file in the Gaussian command. If needed
### more g16 options can be added.

### If using the job composer, you will need to upload your input files to the job
### script's folder. You can do this by clicking on "edit files" and then uploading
### your com file.

#Load Gaussian module
module load Gaussian/g16

#Gaussian scratch directory.
export GAUSS_SCRDIR=/home/$USER/.gaustmp/$SLURM_JOBID
mkdir -p $GAUSS_SCRDIR

#Stop OpenMP from interfering with Gaussian's thread mechanism.
export OMP_NUM_THREADS=1

#Prepare node list for Linda
for n in `scontrol show hostname | sort -u`; do
echo ${n}
done | paste -s -d, > snodes.$SLURM_JOBID

#Run Gaussian. It is recommended to only change the input file here. If needed you can
#raise the memory up to 60GB, but doing so may result in an error.
g16 -m=40gb -p=${SLURM_CPUS_PER_TASK} -w=`cat snodes.$SLURM_JOBID` your_file_name.com

#Clean up nodes list
```

(continues on next page)

(continued from previous page)

```
rm snodes.$SLURM_JOBID
```

---

CHAPTER  
**SEVENTEEN**

---

**JUPYTER/PYTHON**



---

CHAPTER  
**EIGHTEEN**

---

**R/RSTUDIO**





---

CHAPTER  
**NINETEEN**

---

**STACKS**



---

## CHAPTER TWENTY

---

### OVERVIEW

---

**Note:** This section is incomplete. New advanced topics will be added overtime, however if you encounter any issues in the mean time, please contact [administration](#).

---



---

CHAPTER  
**TWENTYONE**

---

**ADVANCED SLURM**



---

CHAPTER  
**TWENTYTWO**

---

**ARRAY JOBS**





## USING GIT AND GITHUB



**MACHINE LEARNING**



---

CHAPTER  
**TWENTYFIVE**

---

**OMPI**



**OURRSTORE**