PetaLibrary and Blanca Training Module

Overview

The goal of this module is to help all INC members who access brain imaging data or run analyses on that data to get acquainted with our Research Computing resources. This module introduces terms that may be new or confusing to a first-time user. To that end, we have included a Glossary as a separate document. Words in this document that also appear in the Glossary will be highlighted. If you are new to data access and analysis, please have the Glossary easily accessible as you read through this document. If you are already experienced with data and analysis, feel free to read this document and only refer to the Glossary on an as-needed basis.

Above all, particularly for new users, we understand that this may be challenging. The best analogy for learning our data and analysis infrastructure is that of learning a new language. Don't be discouraged if there's a term or concept that you don't understand. So often when we learn a new language for the first time, if there's a word that we don't understand in a conversation our brain gets stuck and we stop listening. However, if we just let that word go and try to understand the context and big picture of what's being discussed, we can often guess the meaning of that unfamiliar word by understanding the context. The same applies to this module: try your best to get the ***big picture first*** - how all these pieces fit together. If there are still concepts that are unclear, don't hesitate to reach out!

Research Computing has extensive and excellent documentation on most of the topics we discuss in this module; however, we understand it can be difficult to sift through so much material or even know the right question to ask. In this module we attempt to pull out just the basics that all INC users should know about, including:

- PetaLibrary
- Blanca
- SLURM
- Version control and GitHub
- How to view or access your data
- Basic Linux commands

- Basic Linux permissions
- How to organize data
- Billing structure for data and analysis

Lastly, this module is intended to provide the user with a ***general high-level understanding*** of our data and analysis infrastructure. In this module you will not learn the specifics: those specifics are provided as links to other documents that are more technical. What we do not expect the user will understand after this module:

- Memorization of SLURM commands
- Memorization of Linux commands
- Bash script or programming guru
- GItHub master of the universe

Our goal is for you to grasp the different pieces of the data/analysis language on a broad level, and to know how to look up information or ask questions using the learned vocabulary.

What is PetaLibrary

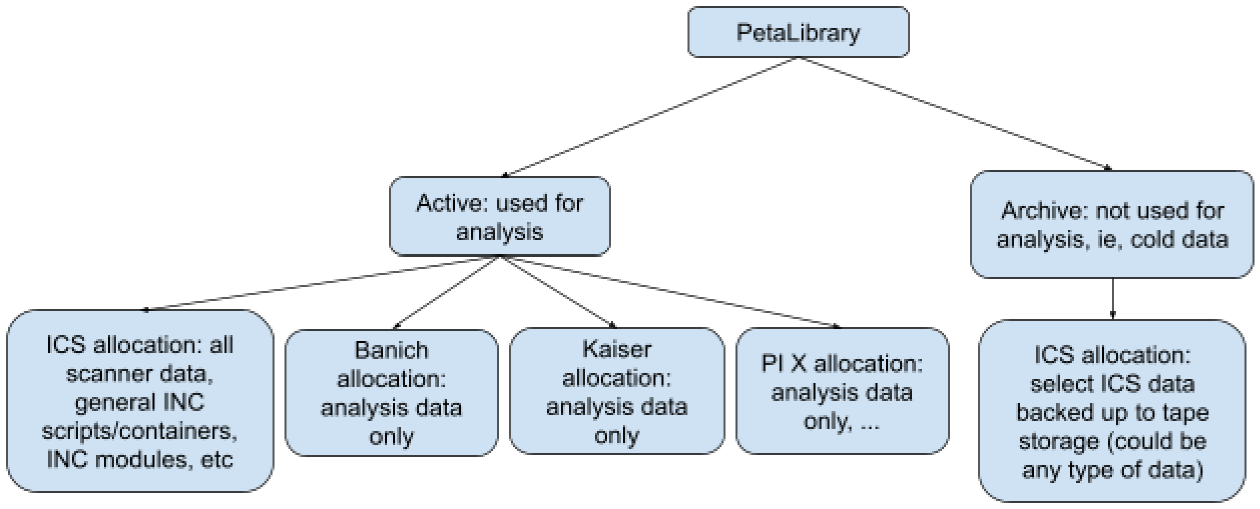

PetaLibrary
Active: used for analysis
Archive: not used for analysis, ie, cold data
ICS allocation: all scanner data, general INC scripts/containers, INC modules, etc
Banich allocation: analysis data only
Kaiser allocation: analysis data only
PI X allocation: analysis data only, ...
ICS allocation: select ICS data backed up to tape storage (could be any type of data)

Figure 1. PetaLibrary and allocation architecture

PetaLibary is the location where data is stored (similar to an external hard drive) and is regularly backed up to ensure data integrity. More accurately, PetaLibrary is a Research Computing managed storage cluster where scanner data and derivative data (i.e. analysis data) live for various projects. PetaLibrary is split up into *active* storage and *archive* storage. For the purposes of this module, we will only discuss *active* storage. If you have questions about or want to request archive storage space, please contact Lena Sherbakov.

PetaLibrary *active* storage is further split up into allocations, which are predefined chunks of space allotted for storage to a given lab or group (eg: banich lab, kaiser lab, etc). All of INC's scanner data goes into an allocation titled *ics*. However, some derivative/analysis data may be most appropriately stored in another allocation that is PI-specific (eg: the *banich* allocation). Figure 1 shows a schematic of the PetaLibrary architecture and allocation breakdown. If you have questions about where to store or access your data (which allocation) *after reading the entire document*, please don't hesitate to contact Lena Sherbakov.

Allocations on PetaLibrary have limits defined in terabytes: for example, our ICS allocation plus the sum of all PI-specific ICS allocations is currently 302 TBs (which we collectively pay to use and maintain), but in the future this number may increase or decrease depending on the need. It is therefore very important to not treat PetaLibrary like you may treat your personal laptop. We are on shared space: adding to one directory on the ICS allocation will decrease the total amount of free space available to everyone else.

What is Blanca?

Blanca is our high performance compute infrastructure, also managed by Research Computing. Most of us are familiar with hard drives, but perhaps less familiar with a computer's processor. While the hard drive is what stores large amounts of data, the central processing unit (or CPU) is what acts on the data (fetches it, decodes it, performs analysis, etc). Figure 2 describes the functions of a CPU; however, to simplify matters when you're first trying to wrap your head around this, think of a CPU as the thing in the computer that does all the thinking!

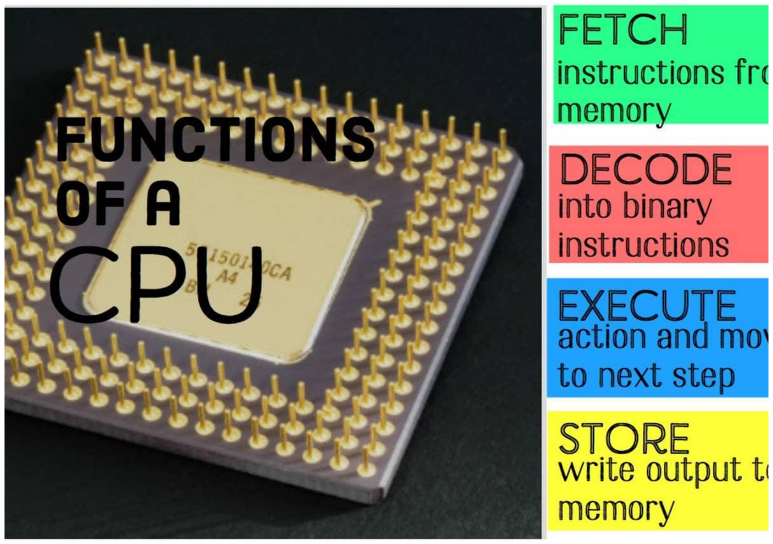


Figure 2. Photo of a CPU and its functions

In simplified terms, a compute infrastructure is nothing more than a bunch of these CPUs and the networking structure that supports them.

As INC, we have 12 nodes (essentially 12 dedicated "computers") on which we can run analyses, collectively known as the *blanca-ics* cluster. Each node has between 28 and 56 CPUs. The reason we put "computers" in quotation marks is because these nodes don't have hard drives, unlike computers. In the analogy above, think of these nodes as the processors, the thinkers, but they can't actually store much information - storage is delegated to PetaLibrary.

INC also has 2 dedicated Blanca login nodes onto which PetaLibrary is mounted/accessible, and through which we can launch compute jobs onto our 12 compute nodes. Figure 3 illustrates how a user interacts with our Blanca nodes.

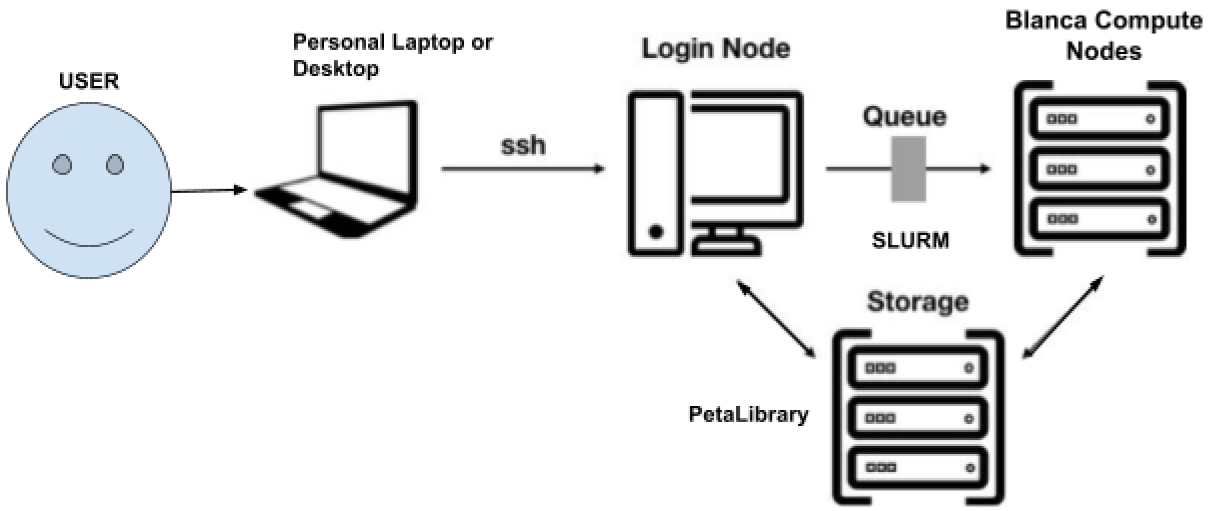


Figure 3. Simplified schematic of data and analysis architecture: a user interacts with his/her own laptop or desktop. That laptop or desktop can interact or log in to a login-node through the SSH protocol (defined later in this document). The login node can then interact with compute nodes (through SLURM commands, defined later in the document). The login node can also read and write data to or from the storage server (PetaLibrary). The compute nodes also have bi-directional access to PetaLibrary.

We cannot emphasize enough: Blanca is not data storage, i.e., it is not where your data lives. However, all of our Blanca nodes have PetaLibrary mounted. An analogy may help: we have all plugged in external hard drives, thumb drives, or flash drives into our computers. The data lives on those drives. However, when we open up our computers, we see that data as if it were a file system on our computers. In this analogy, the external drives full of data are PetaLibrary, while

Blanca is your computer that can open up a document on the flash drive and run some computation.

Types of nodes

Login Nodes

As alluded to above, Blanca nodes can be divided into two types: *compute nodes*, and *login nodes*. As the name implies, login nodes are the access points or login points to Blanca where you can browse the file structure on PetaLibrary, and maybe launch something to view your data. Login nodes will sometimes be called “head nodes” or “blogin” nodes (short for Blanca login nodes). However, ***login nodes should never be used to run compute jobs***.

There are two blogin nodes: blogin-ics2 and blogin01. Both login nodes can be used, although blogin-ics2 is newer. It’s important to know which blogin node you are using because when debugging problems, the INC team or Research Computing will need to isolate the machine you are logged in to, to track down potential problems. [**Advanced user note: the other major difference between the two blogin nodes is the virtual desktop session they run (Xfce on blogin-ics2 and KDE for blogin01). The settings for the KDE desktop on blogin01 are harder to configure; therefore, unless you know what you’re doing or strongly prefer a KDE desktop environment, we recommend using blogin-ics2.]

Both blogin nodes run a Linux Redhat operating system. It is therefore imperative that you are familiar with basic Linux commands (see section below).

Login nodes are a shared resource; many users will be logged on to the same blogin-ics2 node at the same time. As such, what one user does affects everyone else on that login node. For example, if one user is running an application that is taking up all of the processing power of the login node, another user will find it very difficult to do anything on that node (like navigating around directories). Therefore, please be mindful of what you’re running on the login node.

To log in to a blogin node, an SSH protocol (defined later in this document) is used from your local computer. Once you have read through this entire document and are ready to get your hands dirty and try logging in, please review the documentation on logging in here.

Compute Nodes

The Blanca compute nodes, as the name implies, is where we can do all our heavy analyses. Heavy analyses include things like: running neuroimaging tools/programs like FSL, SPM, CONN, or custom python or Matlab scripts. The 12 INC-dedicated compute nodes are heterogeneous, meaning that they are not all identical pieces of hardware (some have beefier processors, some have more CPUs, etc). While the nodes aren't all the same, each node has 256GB of RAM and a minimum of a 2.4 GHz processor (5 nodes have a 2.6 GHz processor). Six of the nodes have 32 processing cores, one node has 56 cores, while the remaining five nodes have 28 cores for a **total of 388 compute cores**.

You may be wondering: why does this detail matter? Well, when we submit a compute job, we specify these requirements (ie, how many of these compute nodes are to be used by the analysis you want to run, how much memory is required, or how many cores or CPUs are desired). For many users, this may seem overwhelming. How do you even know how many CPUs you need? How do you know how much RAM your analysis program needs? The simplest answer to this question is: start small, then scale up if you need to. You can start by comparing the Blanca compute resources to your personal computer. For example, if you're working on a Macbook Pro your CPU will either have 4 or 8 cores. Therefore, requesting that the compute job run on Blanca on a single node (which will have between 28-56 cores depending on the node) is usually plenty.

This answer may leave you unsatisfied. For those wanting to dive deeper into figuring out how to request the appropriate number of resources for your compute job, please read the advanced user training document. For those just wanting to use the default settings, we recommend you run on a single compute node and don't worry about specifying RAM (if the default is not enough, INC staff are happy to walk you through how to increase this limit).

More information on interacting with compute nodes is described in the SLURM section below.

What is SLURM

SLURM is a *workload manager* or *scheduler*: it is an interface that sits between our login nodes and our compute nodes. Think of SLURM as the middle man, broker, or interpreter. To clarify, we'll use the analogy of ordering food from a food truck in a foreign country. As the customer, you'd like to order a grilled cheese sandwich from the food truck. The problem is, you don't speak the language, nor do you know what others have ordered (what the food cart may be out of). You come to the window and thankfully the person taking the orders at the window speaks English! You put in your order for a grilled cheese, and wait while the person taking orders says something to the crew making the sandwiches in their native language. You don't understand what they're saying, but in their native language the person at the window confirms with the sandwich makers that they still have bread, but they're out of american cheese since the customer before you ordered a philly cheese steak. The person taking orders turns to you and now asks in English whether swiss cheese will be okay instead. You have no idea what transpired in the background, but before long you're enjoying your grilled cheese sandwich, with swiss cheese.

In this analogy, the customer is the login node that wants a particular analysis run (grilled cheese sandwich) from the food cart workers, which are the compute nodes. However, the customer doesn't know what other customers may have ordered, what ingredients the food cart may be out of, or how to talk to the food cart workers in a language they understand. Likewise, the login node is oblivious to what compute resources the other users may have requested before you. Nor does a login node know how to talk to the compute nodes directly. In the analogy SLURM is the person taking orders at the window, the broker that helps negotiate this process between what is requested by the customer and what is delivered by the food truck workers (Figure 4).

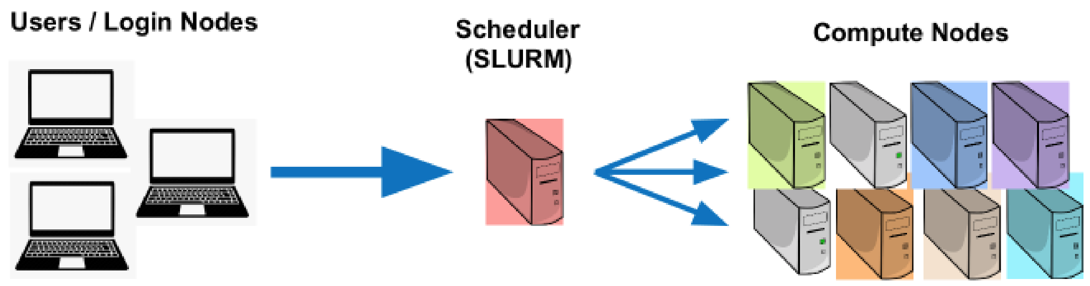


Figure 4: Relationship between Login nodes, SLURM, and Compute nodes

Useful SLURM Commands

All SLURM commands are run from a Linux Terminal. To get familiar with basic Linux commands, please see the Linux section below. There are many more SLURM commands than those listed below. The list below is the absolute minimum a SLURM user should be familiar with. However, the important part of learning SLURM is knowing where to look: nobody is going to memorize all the SLURM commands with all the flags/options that are possible. Rather, the intent here is to inform the user of basic commands and that lengthy documentation exists both on SLURM's website, as well as on Research Computing's website.

The last concept to clear up for new users before talking about SLURM commands is that of a script. A script (aka executable, or program) can be thought of as a file that contains a series of instructions that will be executed by the computer. A script is often helpful when the same set of instructions apply to many subjects that you want to analyze. For example, in the grilled cheese analogy, if you were instead ordering a grilled cheese sandwich for all of your coworkers, it wouldn't be very efficient to sit there and order a grilled cheese sandwich, one by one, 30 times over. Instead, you would simply say in computer-speak: for all the people on this list, I want to order a grilled cheese sandwich.

The most common type of scripts written for computers uses bash (Bash stands for Bourne Again SHell). Think of bash as just another language. Other types of executable scripts include sh (basic shell scripts). Learning to write a script isn't much different from learning basic Linux commands (see section below). Another way to think of a script is as a chain of Linux commands: instead of entering them one at a time, you can write them all down in a document, and when you execute the script, each line of the document will be executed by the computer.

To help the reader associate a concrete use case with the following SLURM commands, throughout the next section we will continue with the analogy of ordering a grilled cheese sandwich, and introduce a more neuroscience-appropriate example: converting raw scanner data (DICOM format) to a more usable format (Nifti format). If you are new to neuroimaging analysis and don't understand why one would want to do this, think of this step as simply converting the scanner data to a format that is recognized by other programs (like converting a .doc file to a .pdf file). The table below summarizes the different SLURM commands and what they do in plainspeak:

SLURM Command	Grilled Cheese Analogy	Neuroimaging Example
sbatch	For all your coworkers, queue up an order of a grilled cheese sandwich. Request that 8 food cart workers work on your order simultaneously. (There may be people ahead of you, and 8 workers may not be available at the moment, but at least you put your order in the queue)	For subjects 1-30, convert dicom formatted raw data to nifti formatted files. The steps for this conversion are defined in a bash script you wrote. Request that 8 CPUs perform the task in parallel (simultaneously).
sinteractive	Request one food cart worker's attention. When you have his attention, ask him to make a grilled cheese sandwich, then another, then another, then another, …	Request an available CPU on any compute node. Once you've been granted access to that compute node, convert dicom formatted raw data to nifti formatted file for

	Until all grilled cheese sandwiches for your coworkers are made, or until your time is up.	subject 1. When that completes, do the same thing for subject 2, then 3, etc, until all subjects are done.
squeue	Ask the person taking orders how many orders have been placed and are currently being worked on. This is useful information if you want to find out how long you'll have to wait for your sandwich.	Request the status from the queue: who is running on what nodes. You may see that all the nodes are currently full, and your job (the conversion task) is pending (waiting for available resources)
sinfo	Ask the person taking orders at the window: "hey what's up with Joe?" (the worker in the back who is on his phone and doesn't appear to be working)	Request the status of a particular node. This is helpful if your job has been assigned to node X but doesn't seem to be completing. Is node X stuck?
scancel	Ask the person taking orders at the window to cancel your grilled cheese. You realize the wait/queue is too long, or you simply messed up, you wanted a turkey sandwich instead.	Request to cancel the script that you submitted with sbatch to convert all subjects from dicom to nifti. You realize there's an error in your script that you want to correct before running all the subjects through.

Now let's get more technical. We've described the commands in plainspeak, but how do you actually launch the commands and with what arguments (ie, with what parameters)? The table below is a more technical example of the basic SLURM commands. You will learn about partitions, qos, and accounts in the next section; for now, just assume they're input parameters or flags to the SLURM commands (for example, specifying a particular worker to make your grilled cheese sandwich).

Command	Usage
sbatch --qos=blanca-ics --partition=blanca-ics --account=blanca-ics-rray --c 8 -J 'fmriPrep' -o 'out_file.txt' -e 'err_file.err' *cmd*	Submit command (*cmd*) to Blanca compute nodes. *Cmd* can also be a bash or shell script. In this example, the user wants to run on queue blanca-ics, partition blanca-ics,

	and sub account blanca-ics-rray. The user is requesting 8 CPUs per task (--c 8), a job name is given (-J ‘fmriPrep’), and an output and error file are defined (‘out_file.txt and err_file.err’, respectively).
sinteractive --qos=blanca-ics --partition=blanca-ics --account=blanca-ics-rray --time=00:10:00	Request an interactive session from a compute node. Instead of submitting a script, the user can interact with a compute node through the terminal directly. If/when an interactive node is granted by SLURM, the user will find their terminal prompt switch from blogin-ics2 or blogin01 to bnode*XXXX*, where *XXXX* stands for the compute node number. This indication on the terminal implies that the user is now directly on a compute node. In the example, the user is requesting a compute node from queue blanca-ics, partition blanca-ics, and sub account blanca-ics-rray for 10 minutes (--time).
squeue -p blanca-ics | grep [idenitkey]	Check status of compute jobs. In this example, the user wants a list of all compute jobs running or pending on partition blanca-ics, filtered for a particular user identiKey (| grep [identiKey])
sinfo -s | grep bnode	Check the status of particular blanca nodes. This command can be used to list all blanca partitions, the states (up, down, drain, idle, etc) of individual nodes in those partitions, and much more. See SLURM documentation. In this example, a summary view (-s) of all blanca nodes (bnode) is requested.
scancel 11791020	Cancel a job running or pending. In this example, the user is canceling his/her job which is numbered 11791020.

For more information on sbatch usage, please refer to SLURM sbatch documentation or Research Computing documentation. Likewise, for more documentation on sinteractive, simply search for sinteractive in the search bar of the Research Computing documentation.

Specifying a Blanca Queue, Partition, and Subaccount

Let’s return to the grilled cheese analogy and make it a little more involved. Let’s assume that all of the food truck workers know how to make a grilled cheese sandwich, but only some of the workers are specifically assigned to customers like you. Other workers take priority orders from other companies (not yours). For example, let's say workers 1,2,3 (call them Team A) are assigned to deal with your orders first; however, workers 4,5,6 (Team B) are assigned to deal with orders from another customer first. Lastly, assume that this priority arrangement for you to use Team A comes with a small up front tip/cost. However, on days when you don't care about how quickly your grilled cheese comes out, you can opt to order without priority, understanding that you may get bumped by orders with priority, but at least you won't pay the upfront tip.

In the SLURM command examples, the two flags or arguments to **sbatch** and **sinteractive** that specify this priority (or no priority), and which Team you want making your sandwich are quality of service (QOS), and partition. *Together with the flag subaccount (discussed next) these arguments are always specified to* ***sbatch*** *or* ***sinteractive****.*

Blanca uses the QOS flag much like a queue or priority designation. ICS users can either enter *blanca-ics* for this field (the equivalent to Team A in the analogy above), or *preemptable* (the equivalent of specifying "no priority" in the analogy above). If qos=blanca-ics, then your job (or grilled cheese sandwich) will take priority on the blanca-ics cluster of nodes. If qos=preemptable, then your job may run preemptively on some other subset of blanca nodes specified with the *partition* flag (ie, not necessarily the purchased blanca-ics nodes. In the analogy above, this would be like specifying Team B to make your grilled cheese sandwich). Running preemptively means that you may get bumped off those nodes if another user (who is not running preemptively) requests the same resources.

The *partition* flag specifies which subset of blanca nodes you want to run on. A list of blanca (bnodes) can be printed by typing: `sinfo -s | grep bnode` in a terminal. Briefly this returns the following list:

Subset of Blanca Partitions
blanca-appm
blanca-ccn
blanca-ceae
blanca-csdms
blanca-el
blanca-geol

blanca-ibg
blanca-ics
blanca-igg
blanca-mrg
blanca-pccs
blanca-rittget
blanca-sha
blanca-sol

Note that INC only pays for the *blanca-ics* partition (12 nodes), so if you'd like to run on any other partition, you need to set the qos to preemptable.

Subaccounts (via the *account* flag) flag is the last required argument to **sbatch** and **sinteractive**. In the grilled cheese analogy above, if you're ordering 30 grilled cheese sandwiches, specifying a subaccount is like writing on the bag full of sandwiches which department in your company the order is for (eg. the HR department). In our case, specifying a subaccount is the equivalent of tagging or associating that analysis job with a particular project or study (eg: adolgaba, rray, abcd, etc).

A complete list of subaccounts of blanca-ics can be gotten with the following command:

```
sacctmgr show account | grep blanca-ics
```

As of August 2021, this command returns the following list of accounts:

Blanca-ics subaccounts
blanca-ics-abcd
blanca-ics-adolgaba
blanca-ics-aim
blanca-ics-cartstrain
blanca-ics-cuwb

blanca-ics-force
blanca-ics-hcp
blanca-ics-ldrc
blanca-ics-ltstwins
blanca-ics-marcog
blanca-ics-most
blanca-ics-nitrite
blanca-ics-npm
blanca-ics-olp4cbp
blanca-ics-paingen
blanca-ics-rmtwins
blanca-ics-rray
blanca-ics-training-only
blanca-ics-ukb
blanca-ics-vrp
blanca-ics-wmem

In order to run an analysis for a particular project, users must contact Lena Sherbakov to add them to the particular sub-account of interest.

If your project or sub-account is not listed above, it can be added by contacting Lena Sherbakov or Amy Hegarty.

For more information on how to use qos, partition, and account flags (with examples) please refer to the documentation on subaccounts written by INC.

Accessing, Viewing, Transferring your Data

There are multiple ways to access, view, and transfer your data from/to PetaLibrary. Below is a description of the appropriate tools to use, and a reference/link to documentation for using those tools. This section is only meant as a high-level overview for users. Linked documents provide more detailed documentation.

SSH

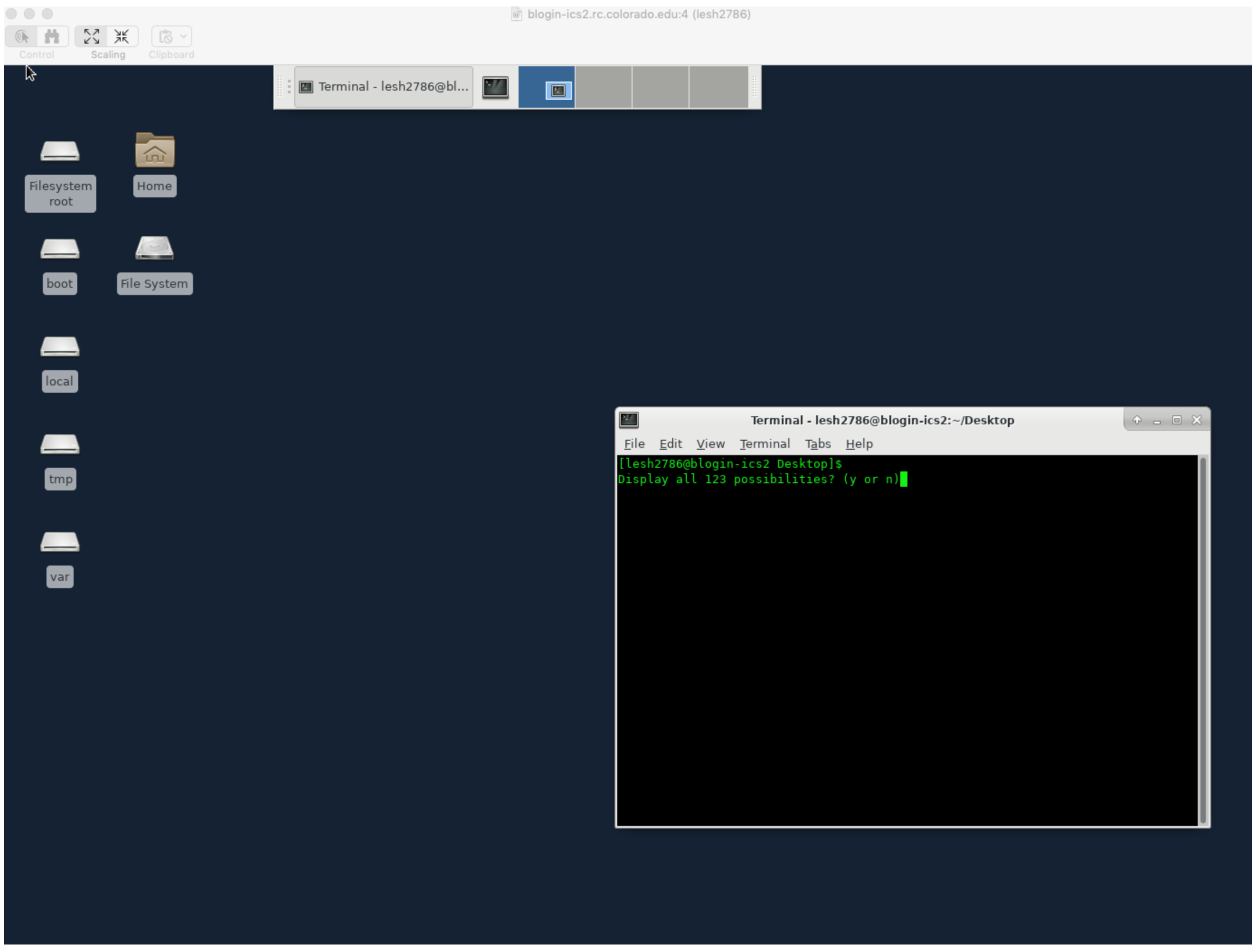
SSH stands for **s**ecure **sh**ell; it is a networking protocol that enables two computers to communicate. If you're new to SSH, think of it as a secure lock and key to a house (where the house is the computer you're trying to log in to). Regardless of what operating system your personal computer runs, SSH facilitates the communication between your laptop or desktop at home and our Blanca login nodes (blogin-ics2, and blogin01) as well as any general Research Computing login node. Like the key and house analogy, more than one user can log in to a login node at a given time (as long as they have the key, ie, the credentials). The house can get so full that 20 or 30 users are all jammed in there; likewise, our blogin-ics2 and blogin01 nodes may have 20-30 users SSH'ed in at a given time. Detailed instructions for how to log in via SSH are provided in the linked documentation.

Once you SSH into a login node from your personal computer's terminal, you'll see the terminal change to: <identikey>@blogin-ics2. The name that appears after the @ sign tells you what computer you have SSH'd in to. However, only a terminal remains - no graphical user interface or desktop pops up for the user. If a user simply wants to list/view the contents of their directories on PetaLibrary, launch a compute job (either through ***sbatch*** or ***sinteractive***), view the progress of their compute job (via ***squeue***), nothing other than SSHing into a login node is needed. From there you can simply type the commands into your terminal window. The user can also edit the content of files, but must do so through shell/terminal based editing programs like vim, nano, etc.

Many users newer to Linux prefer some kind of graphical user interface (GUI) to help them view and edit files (such as sublime, gedit, etc). To view, edit, or launch a GUI, users will need to use either VNC or EnginFrame (described below).

VNC

VNC stands for Virtual Network Computing; it is a graphical desktop sharing system that can be used to view and control a virtual desktop on a remote computer like Blanca. In our case, we can use VNC to launch a virtual desktop (which doesn't look much different than your local computer desktop) on either Blanca login node (blogin-ics2 or blogin01) and connect to it. See Figure 5 to get a sense for what a virtual desktop looks like.


blogin-ics2.rc.colorado.edu:4 (lesh2786)
Control
Scaling
Clipboard
Terminal - lesh2786@bl...
Filesystem root
Home
boot
File System
local
tmp
var
Terminal - lesh2786@blogin-ics2:~/Desktop
File
Edit
View
Terminal
Tabs
Help

```
[lesh2786@blogin-ics2 Desktop]$
Display all 123 possibilities? (y or n)
```

Figure 5. A virtual desktop for blogin-ics2

To use a blogin node with a virtual desktop, you must first launch a vnc ***server*** on the login node of choice (often referred to as your ***remote*** host). The server is what runs the virtual desktop service. To connect to the vnc server running on, say blogin-ics2, you must have a vnc ***client*** on your personal computer (often referred to as your ***local*** machine). Details about how to set up both server and client are linked to in the provided documentation.

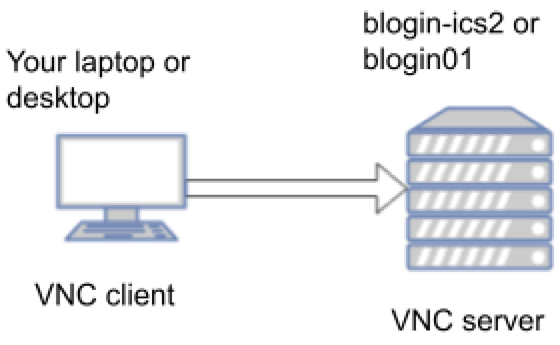


Figure 6. The relationship between a VNC server and client

A common confusion is to mistake a VPN network (Virtual Private Network) for VNC. VPN simply connects you to a remote network (say the CU Boulder network), but it doesn't provide a desktop for you to use. VNC provides remote control of a computer at some other location allowing users to operate that computer as if they were sitting in front of it.

As noted above, where a VNC connection is helpful is if you want to view your data in a non-terminal based app or graphical user interface (ie: fsleyes, sublime, firefox browser). However, VNC problems are not supported by the Research Computing help desk. INC will work with users on setup and debugging of their VNC environments, but it may take time (due to everyone having their own custom configurations on their local machines).

EnginFrame

As an alternative to the remote desktop via VNC, users can access a remote desktop environment through any browser (ie. Chrome, Firefox, etc). EnginFrame is a GPU-accelerated web-based software that enables users to launch a virtual desktop directly from their browser. The pros and cons of EnginFrame as listed below. Documentation for how to log on to EnginFrame can be found here.

EnginFrame Pro	EnginFrame Con
Visualization is faster than VNC with GPU-accelerated remote desktop	Your headnode or login node is a "viz" node, not a blogin-ics2 or blogin01 node. This means not all libraries and modules are automatically loaded for you.
Remote desktop in your browser window (doesn't require setting up VNC server or client)	Desktop is not saved at the close of each sessionm, unlike VNC
Remote desktop is fully configured (doesn't require changing setting of how your desktop looks)	
Research Computing actively supports EnginFrame issues, unlike VNC issues	

SSHFS and SMB Mounting

Both SSHFS and SMB are tools that allow PetaLibrary to be mounted on your local computer, as if it were a file system directly on your local hard drive. To mount something on your computer is like putting a USB stick in your computer. In other words, if you mount PetaLibrary through SSHFS or SMB, you'll be able to see, navigate, modify, copy, and transfer PetaLibrary files through whatever file browser you use locally (ie, on your own computer).

However, mounting PetaLibrary is just that: a connection to the files on PetaLibrary, **not** a connection to the Blanca login nodes, nor any other Blanca nodes. Mounting PetaLibrary through SSHFS or SMB is useful if you want to prototype or profile an analysis on data that exists on PetaLibrary using software, apps, or containers on your own computer. In other words, you would be using PetaLibrary ***data*** but all ***processing*** will be done on your local computer.

As an example, suppose you wrote a new piece of code/software that motion corrects your imaging data. You'd like to debug your algorithm without submitting **sbatch** or **sinteractive** jobs on Blanca, because you're more comfortable with fixing the code on your own computer instead of launching a job, having it fail, tracking down the error files, etc. This use case is a great candidate for using your local computer (where the code exists) for compute but mounting PetaLibrary through SMB or SSHFS to grab the input data.

If your PetaLibrary allocation is already a ZFS allocation (see Lena if you have questions about what this means), you can SMB mount your PetaLibrary allocation to be accessible directly from your local computer. Follow the instructions on the Research Computing page.

If your PetaLibrary allocation is still a BeeGFS allocation, you can use SSHFS to mount your PetaLibrary allocation to your local computer. Follow the instructions on the Research Computing page and don't hesitate to email Lena (lesh2786@colorado.edu) with any questions.

Globus, FileZilla, SCP, Rsync

All of the tools below fall in the category of file transfer tools, ie, moving data between your personal computer and PetaLibrary. These tools do not mount PetaLibrary on your local machine, nor do they facilitate any kind of connection to Blanca login nodes. Instead, these tools are strictly for file transfer.

Globus

Globus is a web-based data transfer tool. It is the easiest way to transfer data back and forth from PetaLibrary because of the user-friendly web interface. Through Globus you can move data to/from PetaLibrary. In order for this to work, you'll need to install a Personal Globus Endpoint on your home computer (or wherever you're trying to transfer data to/from). The Personal Globus Endpoint will make the directories on your home computer visible to the web-based transfer tool and only needs to be downloaded/setup once; if the personal endpoint is running, all future transfers will see your computer and its data in the web-based tool. To set up a Globus transfer, please follow the instructions on the Research Computing website.

FileZilla

FileZilla is another popular tool for transferring data with a user-friendly, GUI-based application. Unlike Globus, a CU Boulder VPN connection must be established before using FileZilla to transfer data to/from PetaLibrary. To set up a FileZilla transfer, please follow the steps on the Research Computing website.

SCP and Rsync

Data can also be transferred via command line/terminal. SCP and Rsync are data transfer protocols; they are used as command line tools (from a unix-like command line Terminal application). SCP and Rsync can be used to transfer data if a GUI is not desirable, or if a shell or bash script has already been written by a user to transfer data in bulk. Globus and FileZilla can also be used to transfer data in bulk, but they are not programmatic, ie. it's a drag and drop operation. The difference with SCP and Rsync is that you can incorporate these commands in a script (that may include other steps of the data transfer process, like creating a new directory, sorting and reorganizing the transferred files). As with the other tools, please read and follow the Research Computing directions for using scp and rsync to transfer data. SCP and Rsync are best used by those comfortable with Linux as they are powerful tools (ie. a lot can go wrong if you're not careful) and no GUI will prevent you from doing something unintentionally to your data (like putting it in the wrong place). Particularly when using rsync, it is recommended to run the command in "dry run" (-n) mode first to catch potential errors in syntax.

Basic Linux Commands

Since our Blanca login nodes run a Linux operating system (Redhat 7), as do our compute nodes, it is ***imperative*** that users understand the basics of a Linux operating system. Linux (unlike Mac or Windows) relies heavily on navigation and manipulation of data and programs from the command line, or Terminal application.

After SSH'ing into a login node, the user will see a blogin-ics2 or blogin01 terminal window. From this terminal window, linux commands can be issued to look at the contents of directories, create files, launch applications, etc. Below is a list of linux commands and shortcuts that users should be familiar with:

Command	Description
`ls -lt <path>`	List files in a directory, where `<path>` is a directory (eg: `/pl/active/ics`). The flags `-lt` display the contents in a list (l) and sort by time modified (t).
`cd <path>`	Change directories to `<path>` (eg: `cd`

	`/pl/active/ics` will put you in the ics directory)
`cp <source file or directory> <destination file or directory>`	Copy a file or directory from the source location to the destination location
`mv <source file or directory> <destination file or directory>`	Move a file or directory from the source location to the destination location
`pwd`	Prints the current/present working directory. Helpful if you get lost/don't know what directory you're in.
`groups <identikey>`	Print all the linux groups that the user with identikey belongs to
`more <file name>`	Prints the content of a text file. If a file has multiple pages, will print one page at a time on the terminal
`cat <file name>`	Also prints the content of a text file but can also be used to concatenate and alter files (must be used with care)
`touch <file name>`	Create a new file with <file name>
`mkdir <dir name>`	Create a new directory with <dir name>
`rm <file name>`	Remove/delete file with <file name>. For removing directories, use the command: `rm -rf <dir name>`
`echo <variable or expression>`	Prints the variable or expression on the terminal. Example: `echo $PATH` will print the contents of the environment variable `$PATH`

`vim <file name>`
`nano <file name>`
`more general launch any application`
	Launch a text editing program called vim or nano and edit a text file.
	More generally, to launch any application, simply type it's name in the terminal

`top`	List all processes running on the current node
`which <name of application>`	Prints full path to application executable. Example: `which top` returns `/usr/bin/top` Useful to determine whether application exists in your known paths
`--help`	This *flag* can be issued after most commands to display a full help menu, ie how to use the

	command
`ctrl-c`	Exit from whatever process or application you started. Useful if you made a mistake in your linux command, the output is taking too long to print, or program is unresponsive
`exit`	Exit from the SSH session

Shortcuts and Special Characters	**Description**
`~`	Home directory. This is `/home/<identikey>`
`.`	Current directory. For example, `ls -lt .` will list contents of current directory
`..`	Parent directory. For example, `cd ..` will change directories to one directory up in the hierarchy, ie, the parent directory. Example: if you are working in directory x/y, then cd .. will put you in directory x
`/`	Path directory separator. For example `ls -lt /pl/active/ics` will list contents of the `ics` directory, whose parent directory is `active`, whose parent directory is `pl`
`#`	When used as the first character in a line, indicates a comment (a command that is not to be executed). Example `#This is a test script`, is a comment that the linux interpreter will not try to execute
`*`	Wildcard. For example, `ls -lt /pl/active/i*` will list all files and directories in the parent /pl/active folder that begin with the character `i`. The wildcard can be used in any character position. Example, `ls -lt /pl/active/ics/*.txt` will list all files in the directory `/pl/active/ics` that have a `txt` extension.
`$`	Variable expression. For example, `my_path=/projects/<identiKey>; echo $my_path` will print out the contents of your variable `my_path` (in this case the string `/projects/<identikey>`)
`;`	Shell command separator. See example above where the two commands (variable declaration and `echo`) are separated by `;`
`&`	Background process. For example, `matlab &` will launch Matlab from the terminal but then return your terminal prompt running Matlab in the background

`<`	Input redirection: redirect input stream from file. Example:
	`while read subject; do`
	`echo $subject; done < list_of_subjects.txt`
	This will read from a list of subjects in the file list_of_subjects.txt and echo out each line, i.e. each subject.

`>, >>`	Output redirection: redirect output stream to a file. Example: `ls -lt /pl/active/ics > ics_contents.txt` will print the contents of the `ls` command to the designated file instead of to standard output, i.e. your terminal window. The double arrow (`>>`) will append the output to the file specified instead of overwriting it.
`|`	Pipe character. Allows users to link multiple commands, ie, the output of the first command becomes the input argument to the command after the pipe character. Example: `squeue -p blanca-ics | grep lesh2786` returns all jobs running on the blanca-ics partition that also contain the identikey lesh2786.

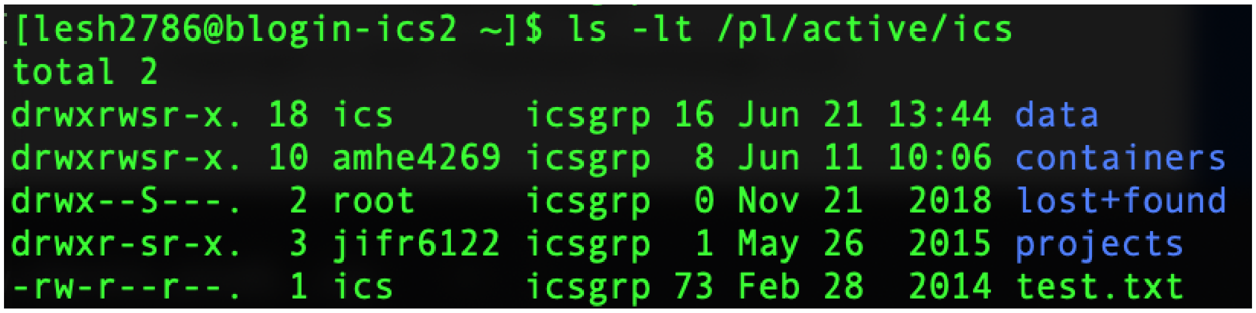
If you would like more linux training, please don't hesitate to contact Lena Sherbakov or Research Computing who can point you to additional resources. Another helpful place to look for linux commands is simply to use Google or StackExchange. For example, if you have a question about how to list only directories, not files, with linux and store the results in a text file, likely somebody else has asked this exact question before and the answer is readily available online.

Basic Linux Permissions

This is perhaps the most important section of the tutorial. Most help ticket questions are a consequence of not understanding the linux permission structure, which is understandable because it is complex and not intuitive for non-linux users.

Each file and folder in linux is tagged with a "user" and a "group". Additionally, different permissions can be given to the user and group that determine who can read, write, and execute that file, script, or directory.

To examine Linux permissions with an example let's consider the output of `ls -lt /pl/active/ics`:



```
[lesh2786@blogin-ics2 ~]$ ls -lt /pl/active/ics
total 2
drwxrwsr-x. 18 ics      icsgrp 16 Jun 21 13:44 data
drwxrwsr-x. 10 amhe4269 icsgrp  8 Jun 11 10:06 containers
drwx--S---.  2 root     icsgrp  0 Nov 21  2018 lost+found
drwxr-sr-x.  3 jifr6122 icsgrp  1 May 26  2015 projects
-rw-r--r--.  1 ics      icsgrp 73 Feb 28  2014 test.txt
```

The output can be interpreted as follows:
The middle section (eg: `ics icsgrp` in the first line of the output) implies that `ics` is the owner of the directory and `icsgrp` is the group. In the same example output above, the second line shows that `amhe4269` is the owner and the group again is `icsgrp`, and so forth. The leftmost set of letters and dashes (eg `drwxrwsr-x`) in the example output above indicates what the owner, group, and any user can do to the file or directory.

Let's consider just the first letter in the first line of the example above (`drwxrwsr-x`): "`d`". This first letter indicates that this entry is a ***d**irectory* (not a file). For the last line in the above example, instead of "`d`" there is simply a "`-`" meaning that the associated entry is *not* a directory (could be a file, executable, etc).

After the "`d`" or "`-`", let's take the entries as triplets. The first triplet to consider in the first row of the above example (`d`**`rwx`**`rwsr-x`) is: `rwx`. These are shorthand permissions for what the *owner* can do. The "`r`" indicates that the owner has ***r**ead* access to view contents in the directory. The "`w`" indicates that the owner can also ***w**rite* into the directory. The "`x`" indicates that if the entry is a script or if there are any scripts inside the directory, the owner can also *e**x**ecute* those scripts. If the owner did not have permissions to do any of these actions, the letter (r, w, or x) would be replaced with a "`-`". Note, deleting data is considered part of writing (since you are *un*-writing), therefore, the user must have `w` permissions to delete data.

The next triplet to consider in the first row of the above example (`drwx`**`rws`**`r-x`) is: `rws`. These are shorthand permissions for what anybody who is a member of the *group* can do. In the example above, the group belongingness in the first line is `icsgrp`. So anybody who is in `icsgrp` will inherit these permissions. Like the owner, we can see that anyone in the `icsgrp` group can both **r**ead and **w**rite into the `/pl/active/ics/data` directory. The "`s`" in place where we should normally expect "e**x**ecute" permissions, stands for "setgid" or **set** g**roup** **ID**. The **s**et group ID ensures that when an executable is run by a user, it will use the privileges of the group. For example, if there was an executable inside the `/pl/active/ics/data` directory, and a user (ex: `lesh2786`) wants to run that executable, it will run with the privileges of the `icsgrp` group (not the `lesh2786grp`).

The last triplet to consider in the first row of the above example (`drwxrws`**`r-x`**) is: `r-x`. These are shorthand permissions for what *anyone* can do to the directory. In other words, anybody (whether they are a member of the `icsgrp` or not) can **r**ead and e**x**ecute data, but they cannot **w**rite into the directory as indicated by the "`-`".

As one can imagine, the topic gets more complicated when we consider nested hierarchical directories (ie. directories within directories within directories, etc) and the permissions therein; however, the above example serves as a good foundation. Lastly, the commands below summarize how one can change the permissions of a file or directory.

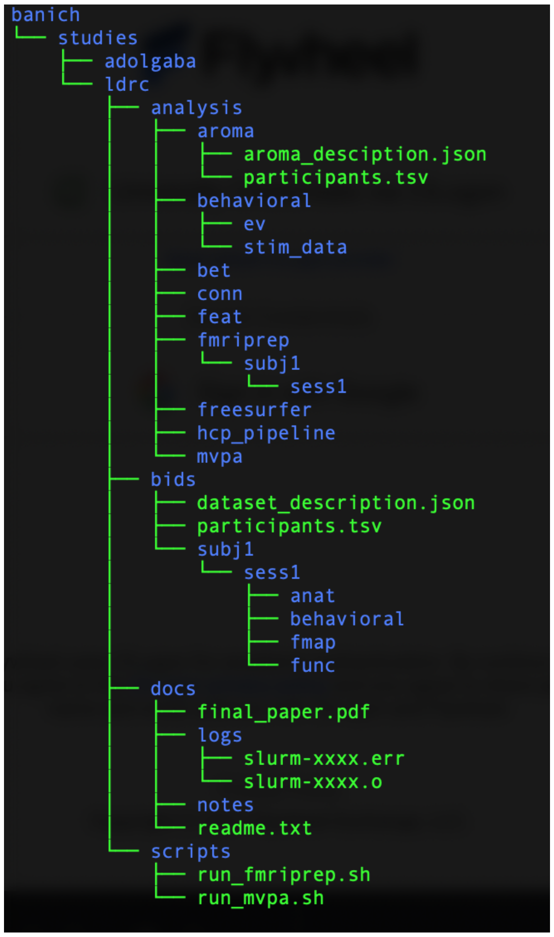
Linux Command	**Description**
`chmod`	This command stands for **ch**ange **mod**e. It allows users to change who has read, write, execute permissions on a file or directory. For example: `chmod g+rwx` will take the group of the directory or file (g) and add (+) permissions to read (r), write (w), and execute (x). This will affect the second triplet in the descriptions above. Example 2: `chmod o-w` will take all **o**thers (not the owner or group), and take away (-) write (w) permissions. This will affect the third triplet in the description above. Example 3: `chmod a+rx` will take all users and give them read, and execute permissions. This will result in all three triplets having `r` and `x` permissions.
`chown`	This command stands for **ch**ange **own**ership. Example: `chown ics /pl/active/ics/containers` will change the ownership of the `/pl/active/ics/containers` directory from `amhe4269` to `ics`. This assumes you have write permissions for both `amhe4269` and `ics` groups.
`chgrp`	This command stands for **ch**ange **gr**ou**p**. Example: `chgrp pl-ics-banich /pl/active/ics/containers` will change the ownership of the `/pl/active/ics/containers` directory from `icsgrp` to `pl-ics-banich`. This assumes you have write permissions on both `icsgrp` and `pl-ics-banich` groups.

Ready to Start a Project? Data Organization

Since PetaLibrary is our shared workspace, we need to keep it organized in a consistent way across the many projects and PIs. If everyone were to use PetaLibrary as their personal Desktop, we would all have a hard time finding files and making sense of the idiosyncratic ways in which each lab organized their data. As an analogy, think of sitting down at someone else's personal computer knowing that you have to find an important document, but all you see in front of you is a cluttered desktop with folder hierarchies that reach 50 layers deep. Where do you even begin to look for the document?

BIDS (Brain Imaging Data Structure) is an effort to do just this: to homogenize how data is stored in an effort to make it more shareable and the analysis more reproducible. If we all had the same conventions about where to put different types of data, we could even write generalized analyses that would work on any study without having to change paths and redefine what files live where. In other words, if I were to write an analysis script and all my BIDS data lived in `/pl/active/kaiser/studies/rray/bids`, to apply the same script to another study (eg. adolgaba), all I would need to do is change the name of the study (eg: `/pl/active/kaiser/studies/adolgaba/bids`). However, if the adolgaba study used a completely different directory structure, I would have to change the whole path and hunt that difference down throughout the code.

While the neuroimaging community hasn't quite nailed down the final BIDS structure for derivative (analysis) data, we (INC) are adopting a BIDS-like storage structure now before data volumes grow out of control. Below is a sample directory schematic for how we would like all studies to organize their data. In this sample schematic, the PI allocation is `banich` and the exploded tree structure is provided for the `ldrc` study.



```
banich
└── studies
    ├── adolgaba
    └── ldrc
        ├── analysis
        │   ├── aroma
        │   │   ├── aroma_desciption.json
        │   │   └── participants.tsv
        │   ├── behavioral
        │   │   ├── ev
        │   │   └── stim_data
        │   ├── bet
        │   ├── conn
        │   ├── feat
        │   ├── fmriprep
        │   │   └── subj1
        │   │       └── sess1
        │   ├── freesurfer
        │   ├── hcp_pipeline
        │   └── mvpa
        ├── bids
        │   ├── dataset_description.json
        │   ├── participants.tsv
        │   └── subj1
        │       └── sess1
        │           ├── anat
        │           ├── behavioral
        │           ├── fmap
        │           └── func
        ├── docs
        │   ├── final_paper.pdf
        │   ├── logs
        │   │   ├── slurm-xxxx.err
        │   │   └── slurm-xxxx.o
        │   ├── notes
        │   └── readme.txt
        └── scripts
            ├── run_fmriprep.sh
            ├── run_mvpa.sh
```

Of note, this structure does not include duplicating the raw scanner data (the dicoms). These dicoms live in our `ics` allocation (eg. for the ldrc study, they are here: `/pl/active/ics/data/archive/human/dicom/prisma_fit/mbanich/ldrc_200122`). The directory structure includes bids converted data (in the example above, sub-directory `/banich/studies/ldrc/bids`). Each analysis that is run on a dataset is in its own subdirectory. Scripts and code to run the analyses are also in their own directory that can be separately version controlled (*scripts;* see section below). `Json` and `tsv` sidecar files are used to describe each analysis and list of participants for that particular analysis. A `readme.txt` file exists that can be generalized to a spreadsheet which keeps track of analyses run, parameter configurations, results, etc. The above directory structure is just a skeleton. Other sub-directories may and will likely exist for study specific setups (for example: if your study includes analyses for three separate FSL Feat models, those analyses would exist in subdirectories under the `feat` directory). However, if we can keep the high level structure the same, we're already halfway there to navigating around the data.

Most importantly, what we want to avoid in data structures are:

- duplicating data (if you absolutely need your dicoms in this structure, at least soft link them instead of creating another hard copy)

- nondescript directories like tmp/temp, which don't give an indication of what is inside those directories
- very deep directory structures that keep the user drilling down indefinitely (eg. /data/a/b/c/d/e/f/g/...)
- scripts that are tangled with analysis directories (eg. scripts that are inside analysis directories). The reason this is a problem is because it is not easy to separate those scripts and version control them, while leaving the data out of the version control system. Fore more on version control, see section below.
- multiple failed versions of the same analysis (instead, note down the parameters, what worked/didn't work, and delete the failed version). For example, if you decide to use a spatial smoothing parameter, x, but after looking at the results see that the smoothing is way too aggressive, it's best to note that in a text or excel file and delete this failed/overly-aggressive smoothed data.

As always, if you have any questions about best practices of organizing your data, please reach out to Lena Sherbakov or Amy Hegarty.

GitHub and Version Control

The best way to maintain your code and analysis scripts is to store them on a version controlled system like GitHub. GitHub allows the user to track changes to code, scripts, or regular files without creating duplicate copies. Git is also a great way to securely collaborate with others on code without worrying about who is overwriting changes to what file. For example, if two people make edits locally to the same file, before that file can be "pushed" to the code repository via Git, the merges and merge conflicts will be explicitly called out and reconciled (eg: Git will send a message like "Lena changed line x of script, but Amy also changed line x of script, here are the two versions, what would you like to do?"). Lastly, Git can be used to synchronize your remote code repository that exists online with a local copy of the code in your PetaLibrary `scripts` directory, for example.

Learning to use Git or any other version control software is a topic worthy of its own tutorial. However, users should at the very least be aware of what Git and other version control software do. For more information, or to request a Git tutorial, please contact Amy Hegarty.

Why is This Important: Data and Analysis Rates

The reason we are stressing this tutorial on our users is because:

1) We store data and process data on a shared infrastructure. What one user does impacts the whole community
2) Prudent data processing and storage will avoid excess data/analysis charges. For an up-to-date description of INC's rate model, please see the Facilities and Services tab on our website
3) We're a University! We believe that education is at the forefront of our responsibilities to our users. Other centers may hire out this work and never teach their students; however, we believe that everyone benefits from learning the basics of how to store and analyze your research data.

Where Do I Get Help

We understand that this tutorial, while just the beginning, covered a lot of ground. The most important takeaway from this module is knowing where to look for additional help/information. Below is a list of data and analysis resources by CU group.

INC Resources

Please do not hesitate to reach out to Lena Sherbakov (Data Scientist at INC) or Amy Hegarty (Neuroimaging Data Analyst at INC) with follow up questions. We also host Data/Analysis office hours - please check the MRI calendar for times. You're welcome to let us know you're coming or just drop in to the office hours with your questions.

We (INC) will also be starting trainings/tutorials at least once a quarter during the school year. Feel free to email or talk to us about what you'd like to see for the contents of those trainings!

Research Computing Resources

Research computing is a great resource and very supportive of our INC community. If the question you have is specific to Research Computing resources, please contact rc-help@colorado.edu.

Note: while research computing will help you with most of your questions, they do not support VNC and VNC-related issues. If in doubt, feel free to send Lena/Amy an email first, and we're more than happy to contact Research Computing on your behalf if we can't answer your question.

Quiz

Now that you have finished the Peta Library and Blanca training Module, please take the quiz! You can retake the quiz as many times as you like to get a passing score.

Additional Courses and Useful Documents

- INC website
- Research Computing webpage with documentation
- CRDDS courses