# Intro to using ITTC cluster computing for Walters Lab Members

To register for an account on the ITTC cluster:

<https://secure.ittc.ku.edu/account/>

Jamie or ClusterAdmin can provide username & password. (Jamie has it in Evernote).

Many details can be found on the ITTC user reference wiki website:

<https://help.ittc.ku.edu/Cluster_Documentation>

For help navigating or managing cluster resources, you should email: clusterhelp@ittc.ku.edu

Emailing ClusterHelp is also the best way to request the installation of additional software.

1. Logging on to remote server
   1. Use ssh
   2. On campus, can ssh directly to login[1|2].ittc.ku.edu

ssh <userid>@login1.ittc.ku.edu

* 1. From off campus, activate KU CiscoConnect VPN, then login as if on campus
  2. It is possible to login from off campus without KU VPN.
     1. First connect to the ITTC ssh server (use port 62)

ssh -p62 <userid>@ssh.ittc.ku.edu

* + 1. Second, from the ssh server, you can connect to login servers as above.

1. Transferring files
   1. scp on mac/linux – this is command line, though GUI clients exist

**To push a file from local to server:**

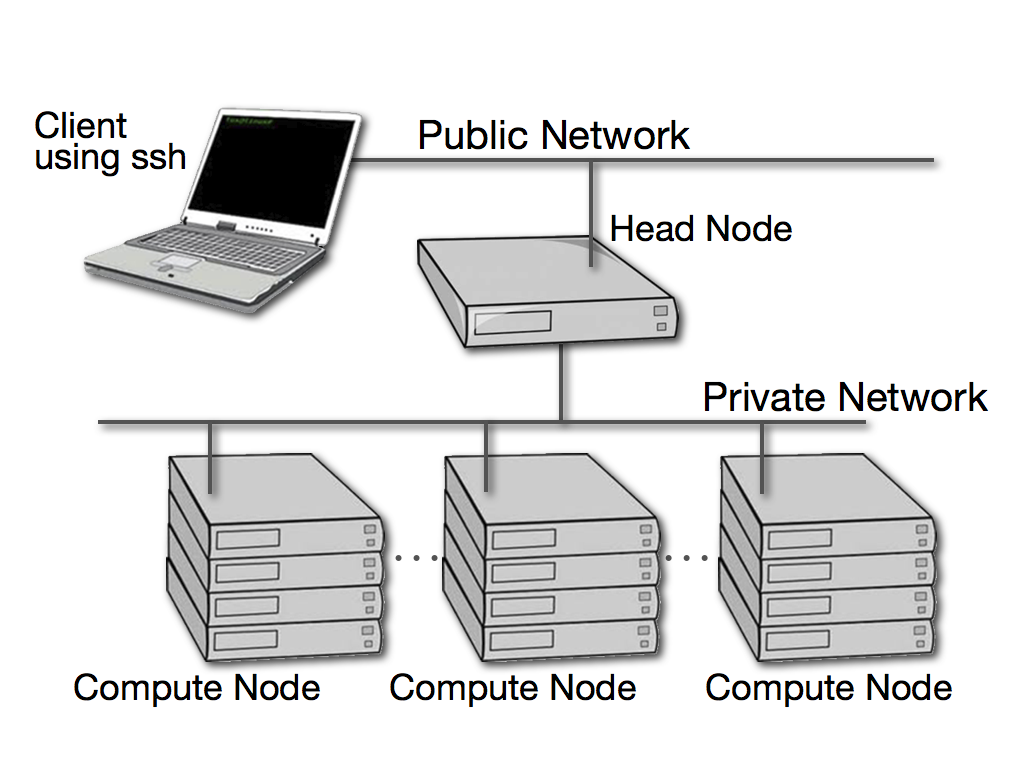
scp myfile.txt <user>@transfer.ittc.ku.edu:<path>/myfile.txt

**To pull a file from server to local:**

scp <user>@transfer.ittc.ku.edu:<path>/myfile.txt myfile.txt

* 1. WinSCP on windows – this is GUI. Putty gives DOS command line if desired.

1. [Filesystem](https://help.ittc.ku.edu/Cluster_Documentation#Filesystems) directories
   1. Home: /users/<userid>
      1. This is you default entry point every time you log on to the server. There is 5Gb quota of disk space on this directory, so do not store much data here. This space is primarily for configuration files, script files (e.g. your own ~/bin file), etc.
   2. Work: /work/jwalters/<uderid>
      1. Use this space to store data/scripts that are routinely re-used in various analyses, or is otherwise not considered “temporary” or easily replaced/reconstructed
      2. Everyone in the lab has their own work directory under my name and there is a single combined quota across all group members (in contrast to scratch, where user dirs are not grouped under my name and the quota is individual)
      3. In /work, all directories and files are accessible by everyone in the group. This facilitates sharing data and files.
      4. Available file storage (the quota) is 1 Tb, but this is SHARED for all Walters-lab group users.
   3. Scratch: /scratch/<userid>
      1. Scratch is where you will do most of your work, since it has very high quotas and is generally meant to be for “temporary” storage.
      2. Data are not routinely purged or wiped. Rather, it is the user’s responsibility to limit total usage below the allowed quota.
   4. RFS (aka Research File Storage) /rfs/jwalters/EXPORT
      1. RFS is used as archival file storage. It has tape back-up. The primary purpose of this directory is to store primary data that along with key elements important analyses cannot be (easily) reproduced.
      2. RFS is not connected to the cluster with “high speed” access. You should not run analyses on the cluster by reading data directly from RFS. Always copy your files to /work or /scratch first, which have high-speed I/O connections to the nodes.
      3. Never delete or overwrite anything on /rfs/jwalters without asking permission from Jamie
      4. This drive is also connected to the CRC cluster, so can serve as a bridge between the two systems if need be.
   5. TMP: /tmp -- local file storage on compute nodes
      1. On rare occasions it might be useful to write very temporary files to the local compute node, with will be accessed as /tmp
   6. Checking your quota with “myquota”
      1. If you need to know how much of your allotted data storage is available, use the command “myquota”. If you go over your allotment you will no longer be able save/write files.
   7. Storage backups:
      1. Home (i.e. <users>): Backed up to tape nightly, with a 3-month history.
      2. /work: 30 days of disk snapshots
      3. /scratch: 7 days of disk snapshots
      4. access the snapshots through the hidden .snapshot directory (for example: /scratch/.snapshot)
2. Modules: What software is installed?
   1. The ITTC cluster provides a vast array of software already installed. However, you must first load it into your environment using the “modules” framework.
   2. Use ‘module avail’ to list the installed software
   3. Use ‘module load <module/version>’ to import the software into you current working environment
   4. Practice this: add these programs to your environment
      1. BLAST+ v2.6
      2. Samtools v 1.3
   5. You must do this every time you initiate a new session on the cluster, including inside SBATCH submission scripts (see below).
      1. For regularly used software, you may want to include ‘module load’ statements in your profile so they are loaded by default.
3. Understanding computer clusters:
   1. Head vs Compute nodes (aka Boss vs Worker)



* 1. Compute nodes have different capacities:
     1. Number of cores
     2. Available memory
     3. Processor architecture
  2. Job management & priority is controlled by software: **SLURM**
     1. SLURM allows detailed specification of
        1. nodes to use for your job
        2. job queues
     2. SLURM balances user requests to maintain fairness

1. Job queues
   1. “scripted queues” – you submit jobs as completed scripts
      1. some further details for [ITTC queues are here](https://help.ittc.ku.edu/Cluster_Documentation#Cluster_Partitions)
      2. The queues are organized by processor type, with ‘intel’ as the default. However, ‘amd’ nodes are just fine for our jobs, so if ‘intel’ is oversubscribed and your jobs aren’t starting quickly, try submitting to ‘amd’.
   2. The “interactive” queue allows direct login to nodes.
      1. Use ‘slogin’ command
         1. slogin is a convenient wrapper around [the ‘srun’ command](https://help.ittc.ku.edu/Cluster_Documentation#srun), which allows specific node configurations to be request.
      2. Good choice for testing your jobs.
2. Tmux – maintain interactive session while not connected.
   1. There are MANY tmux tutorials, just google for them.
   2. Best practice for interactive queue:
      1. Start tmux session from login (head) node
      2. Start interactive session from inside tmux
   3. There is a tmux cheatsheet in the github repo, FYI.
3. Submitting scripted jobs: using **sbatch**
   1. ‘sbatch’ submits scripts to queues.
   2. sbatch scripts are just normal shell scripts, with a few additional specially formatted comment lines that give run parameter details to SLURM.
   3. The ITTC wiki gives [some examples](https://help.ittc.ku.edu/Cluster_Documentation#sbatch), and the [complete sbatch documentation](https://slurm.schedmd.com/sbatch.html) is available.
4. Monitoring job status and managing jobs
   1. There are a [handful of commands](https://www.rc.fas.harvard.edu/resources/documentation/convenient-slurm-commands/) for monitoring and modifying/killing jobs in the queue.
   2. The ITTC site gives [this summary of useful Slurm commands](https://help.ittc.ku.edu/Cluster_Documentation#Helpful_Commands).

Let’s practice using the cluster:

First, clone the repo into your /scratch dir.

> git clone https://github.com/WaltersLab/ITTC\_Cluster\_Intro.git

Then run the ‘simple\_hw.sh ’ script

> bash simple\_hw.sh

Now start (or attach) a tmux session, start a new interactive session on a node, and run this on a node.

> tmux attach

> bash simple\_hw.sh

Now exit/quit from the interactive node session and disconnect from tmux (exit, then ctrl-b d)

Now submit the script to qsub, then check status with squeue

> sbatch simple\_hw.sh

> squeue –u <userid>

Where is the output? It does not print to the terminal. Jobs run on compute nodes via SBATCH, by default, write STDOUT & STDERR to slurm-<jobid>.out

[jwalters@login2 practice\_scripts]$ ls -lh

total 144K

-rwxrwxr-x. 1 jwalters jwalters 293 Jun 21 13:57 simple\_hw-sbatch.sh

-rwxr-xr-x. 1 jwalters jwalters 188 Jun 21 13:40 simple\_hw.sh

-rw-rw-r--. 1 jwalters jwalters 124 Jun 21 15:17 slurm-166635.out

Now examine the script simple\_hw-sbatch, which has the additional SBATCH commands added:

#!/bin/bash

#SBATCH -J HloWorld

#SBATCH -o HloWorld-%j.out

#SBATCH -e HloWorld-%j.err

#SBATCH -p intel

#SBATCH -n 1

#SBATCH -c 1

#SBATCH --mem=1GB

Copy Here we named the job, as well as specified different files for output and error streams, which is often helpful. Other details include the number of nodes (-n), cores (-c), memory (--mem) and the queue (-p)

Note that there is flawed request to change directory, so that there is an error message written to the error file.

IMPORTANT: you can always run your script with SBATCH specifications as a ‘normal’ shell script. All of the #SBATCH are simply ignored as comments. Just make sure you have logged into an interactive session (slogin or srun) that emulates the requirements specified in your SBATCH arguments (i.e. requested cores, memory, etc.). This is very useful for testing that your script will work as intended.

# PRO TIPS!

**SBATCH: -N vs –n vs –c** (from Wesley Mason)

If you aren't using MPI and only use one node for a job, then you can ignore

the "-N" and "-n" options in favor of only using "-c". The "-n" tasks option is

used to create environment variables used by MPI to launch the correct number

of processes/tasks. The "-N" and "-n" options are really only helpful when you

need precise control on how your MPI jobs launch across multiple nodes. If you

specify the number of cores to use when you launch your program in your job

script (ex: blastall -a 4 or blastn -num\_threads 4), then you will only need to

match that with the "-c" option.

To elaborate, I think of the "-c" option as the number of processes/threads I

want to launch on one node. I think of the "-n" option as the number of

different instances of the executable being launched under MPI.

**One-liners using “wrap” with sbatch**

Consider a situation where you have several simple jobs you want to run in parallel by submitting to the queue. These jobs are arguably too simple to deserve a “stand alone” for shell file, for instance sorting several bam files in parallel. In other words, you are in the world of “one-liners”. You can do this with the “wrap” argument, which gives an “anonymous” command to be treated as a shell script. It would look something like this:

samples='RAF1\_CPOM\_OGS3 RAM1\_CPOM\_OGS3 RAF2\_CPOM\_OGS3 RAM2\_CPOM\_OGS3 RAF3\_CPOM\_OGS3 RAM3\_CPOM\_OGS3'

for i in $samples

do

sbatch -N 1 -c 4 -J $i --mem 8G --wrap="samtools sort -m 1.5G -o $i.sorted.bam -T /tmp/$i.sorting --threads 4 $i.0\_genomic\_unmasked.bam"

sleep 1

done

The idea here is use a bash loop over sample names and letting sbatch read the command directly, rather than having to write a different shell script for each sample. This will launch a separate queued analysis for each sample.

**Local node storage**

In some cases it is useful to be able to store data locally on the working node, rather than in /scratch or another networked directory. For instance, this is useful when the software creates many temporary files. In this case, you can specify the local node’s hard drive by as the /tmp directory. Typically this will give you in the range of 100-200 Gb of storage space to work with, assuming nobody else is writing to the same node, or there aren’t lingering files (like after a run crash) that weren’t erased.

In addition, you can use the local node’s RAM as storage, which will provide very fast access to such files. This is accessed as /dev/shm. Note that the amount of RAM storage will depend on the node’s RAM.

**Modifying file permissions**

Typically the most straightforward way to share files is to move them into a directory in /work, where everyone in the group can access all folders. However, in some cases it may be preferred to simply change the permissions on the folder files themselves so others can read (or write or view). Here is a good, relatively succinct but [informative tutorial on Linux permissions](http://catcode.com/teachmod/). However, in most cases the thing you’ll want to do is make files readable for others in the walterslab “group”. Here is an example, from Wesley, of making files group readable in /scratch:

You can accomplish this by setting the group ownership to the "hpc\_walters" group and allowing group read&execute on directories and group read on files. The tricky part is you need to allow this for all parent directories too. Your parent directory "/scratch/jwalters" only allows user read&write&execute, so you would need to run the following commands to allow group read access to the subdirectory Heliconius\_genomic\_fastq:

$ chmod g+rx /scratch/jwalters

$ chgrp hpc\_walters /scratch/jwalters

$ chgrp -R hpc\_walters /scratch/jwalters/Heliconius\_genomic\_fastq

$ chmod g+rx /scratch/jwalters/Heliconius\_genomic\_fastq

$ chmod -R g+r /scratch/jwalters/Heliconius\_genomic\_fastq