

SMITHSONIAN HIGH PERFORMANCE COMPUTING

Hydra

User documentation for Hydra, the Smithsonian Institution High Performance Computing cluster

Generated 2026-10-06. The current version is at <https://computing.si.edu/hydra/>.

Contents

Getting Started

Getting Started	4
Requesting an account	5
Logging in and passwords.....	6
Training and workshops	11
Quick start	12

Running Jobs

Running Jobs	15
How the scheduler works.....	16
Write and submit a job.....	20
Request a queue, memory and CPUs.....	25
Submit a job array	29
Submit a parallel job	33
Monitor and manage jobs	38
QSub Generator	43
Queues.....	46
Job script reference	50
Resource limits.....	54
Monitoring tools	57
Warning emails	63
Examples	66
Cluster hardware.....	67

Interactive Use

Interactive Use.....	70
Start an interactive session	71
Jupyter	73
VS Code	75
RStudio server	79
RStudio on a compute node.....	86

Storage

Storage	88
Filesystems	89
Check usage and quotas.....	91
Recover a file from a snapshot	94
Scrubbed files and restores	96
Use a node's local SSD.....	98
Use /store and the I/O queue	101
Backups	103

Software

Software.....	104
Find and load software.....	105
Python and conda	107
R	111
Julia, MATLAB, IDL, Java	113
GPUs	116
Containers	119
Write a module file.....	120
Compilers, libraries and MPI.....	122
Installed modules.....	125
Software guides	134
BEAST	135
BirdNET	137
BLAST	140
Blast2GO.....	142
BUSCO	144
MaSuRCA	145
SMC++	147
SPAdes	148
Local databases	150

Data Transfer

Data Transfer	152
scp, sftp and rsync.....	153
Globus.....	160
rclone and cloud storage	167

Policies & Support

Policies & Support.....	170
Usage policies	171
Citing Hydra	174
Cluster status.....	175

Getting Started

If you are new to Hydra, work through these four pages in order.

1. [Request an account](#)
2. [Set your password and log in](#)
3. [Watch the required training video](#)
4. [Submit a first job](#)



Hydra's compute nodes at the Ashburn data center. (Image modified using AI to enhance color grading and reduce image noise.)

If you have used a Slurm cluster elsewhere, expect different commands. Hydra runs Grid Engine, so you submit with `qsub` and check with `qstat` rather than `sbatch` and `squeue`. [How the scheduler works](#) explains the differences.

When Hydra contributes to a publication, please [cite it](#).

Last updated 2026-10-03

Requesting an account

Anyone with an active Smithsonian network account can request a Hydra account. The request goes through the SI Service Portal, and we create the account, usually within a few working days.

A Hydra account is separate from your SI, CfA and VPN accounts

We set the username and password ourselves. They are not synchronized with any other SI system, so changing your SI password does not change your Hydra password.

Request the account

Submit the [HPC Account Request](#) in the SI Service Portal, under the Accounts & Access catalog. The form asks for your name, your unit, your supervisor's name and approval, and a sentence or two on what you plan to run.

When the account is ready you receive an email with your username and a link to set your initial password. The username is normally your SI email name, in lower case.

Request VPN access if you work off site

The login nodes accept connections only from the Smithsonian network, the SI VPN, or telework.si.edu. If you will work from outside SI, also submit a [VPN account request](#). Telework needs no extra account and works from anywhere, but it gives you a terminal in a browser rather than a full ssh connection, so most people who work off site want the VPN as well.

When the account is ready

Set your password from the link in the welcome email and go to [Logging in and passwords](#). Before you run a job, watch the [introduction to Hydra](#) video. We ask this of every new user. It covers the [usage policies](#), which you agree to by using the cluster.

Last updated 2026-10-03

Logging in and passwords

You reach Hydra through an **ssh** connection to one of its two login nodes, `hydra-login01.si.edu` and `hydra-login02.si.edu`. Either one will do. They share your home directory and see the same cluster. From a computer on the Smithsonian network or the SI VPN you connect with an ssh client. From anywhere else you use the web terminal on `telework.si.edu`.

Where you can connect from

The login nodes accept connections only from:

- a computer on the Smithsonian network,
- a computer connected to the SI VPN ([VPN authentication instructions](#)), or
- the web terminal on [telework.si.edu](#), which works from anywhere.

A connection from anywhere else is refused before you are asked for a password.

Log in

Three failed logins lock the account for 15 minutes

Wait the 15 minutes before trying again. Retrying during the lockout restarts it.

Replace `USERNAME` with your Hydra username from your welcome email, in lower case.

macOS and Linux

Open a terminal (on macOS, Terminal is in `/Applications/Utilities`) and run:

```
ssh USERNAME@hydra-login01.si.edu
```

The first time you connect, ssh asks you to confirm the host key. Type `yes`. It is asking whether you trust the computer at the other end, and after the first time it remembers the answer.

```
The authenticity of host 'hydra-login01.si.edu' can't be established.  
RSA key fingerprint is ...  
Are you sure you want to continue connecting (yes/no)? yes
```

At the `Password:` prompt, type your password. Nothing is echoed while you type, not even dots.

Windows

Windows 10 (version 1803 and later) and Windows 11 include ssh. Open Command Prompt or PowerShell from the Start menu and run:

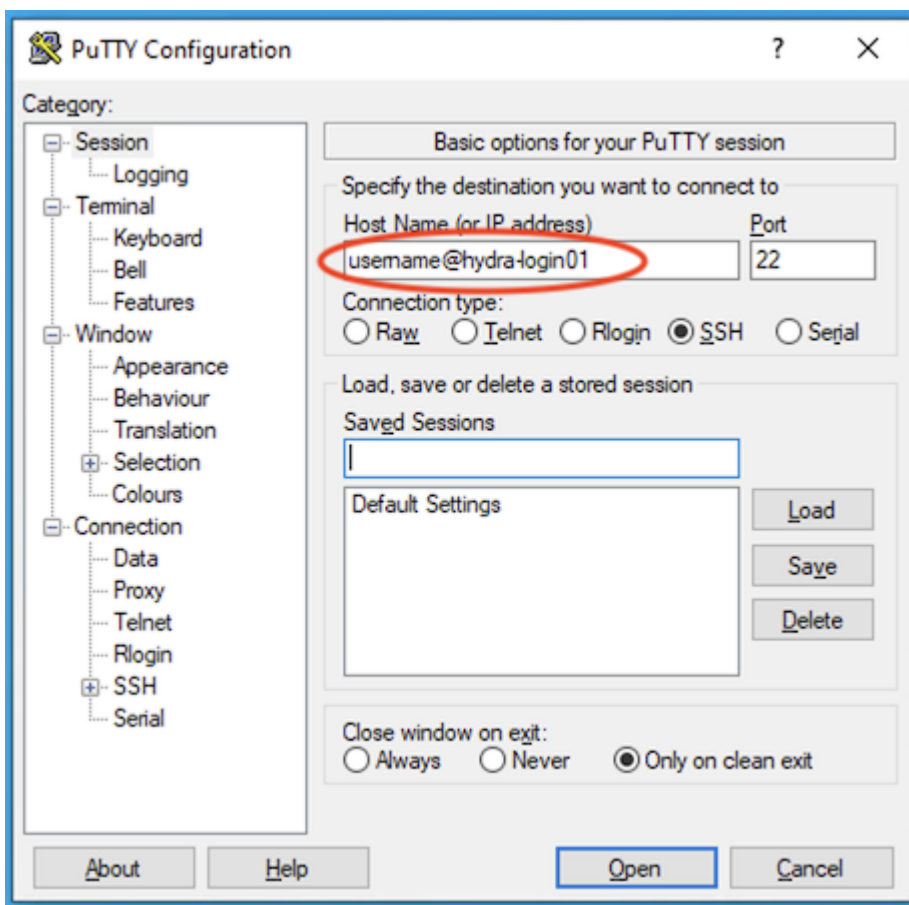
```
ssh USERNAME@hydra-login01.si.edu
```

The first time you connect, ssh asks you to confirm the host key. Type **yes**. It is asking whether you trust the computer at the other end, and after the first time it remembers the answer.

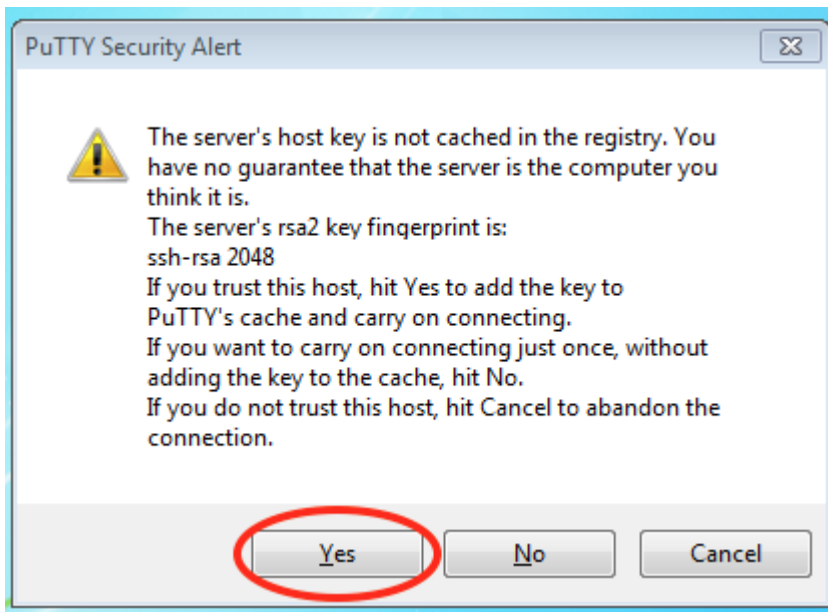
```
The authenticity of host 'hydra-login01.si.edu' can't be established.  
RSA key fingerprint is ...  
Are you sure you want to continue connecting (yes/no)? yes
```

At the **Password:** prompt, type your password. Nothing is echoed while you type, not even dots.

If instead you see 'ssh' is not recognized as an internal or external command, use **PuTTY**. Download **putty.exe** (no installer needed) and start it. Enter **USERNAME@hydra-login01.si.edu** as the Host Name and click **Open**:

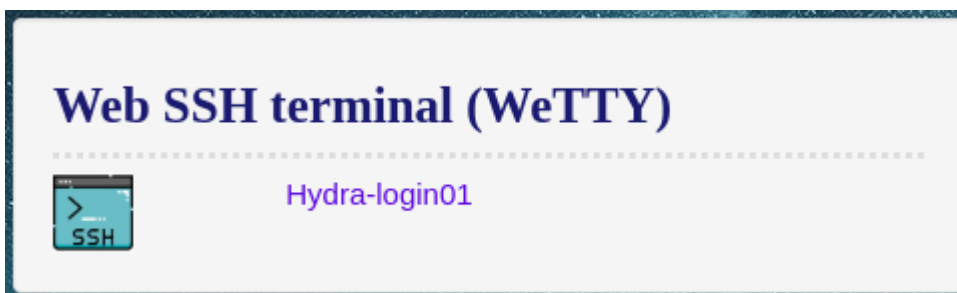


The first time, PuTTY shows a security alert about the server's host key. Click **Yes** to accept it. PuTTY then remembers it.



Web browser (telework)

1. Sign in at telework.si.edu.
2. Expand **IT Tools** and choose **Hydra** (type `hydra` in the search box to find it).
3. Choose one of the **Web SSH terminal (WeTTY)** links.
4. Enter your Hydra username at the `login:` prompt and your password at the `password:` prompt.



You are logged in when the prompt changes to:

```
[USERNAME@hydra-login01 ~]$
```

Log in without a password

On macOS and Linux you can use an **ssh key pair** instead of a password. The pair is a private key that stays on your computer and a public key that you copy to Hydra. Generate a pair with `ssh-keygen` (accept the defaults, and give the key a passphrase), then copy the public key to Hydra:

```
ssh-copy-id USERNAME@hydra-login01.si.edu
```

The home directory is shared by both login nodes, so one copy serves `hydra-login02.si.edu` too. From then on `ssh`, `scp` and `rsync` to either node ask for the key's

passphrase, or for nothing if your computer's keychain holds it, rather than for your Hydra password.

Passwords

You set your initial password from the link in your welcome email. A password is valid for 180 days, and Hydra emails you before it expires. Change it before then, either with `passwd` on a login node or on the self-service page. The self-service page also resets a password you have forgotten, and one that has expired, for 14 days after the expiry date. After those 14 days the account locks and only we can reset it.

Password requirements

A password has at least 12 characters, with at least one digit, one upper-case letter, one lower-case letter and one special character. A new password must not be similar to a previous one.

Change your password from the command line

While your password is still valid, log in and run `passwd`:

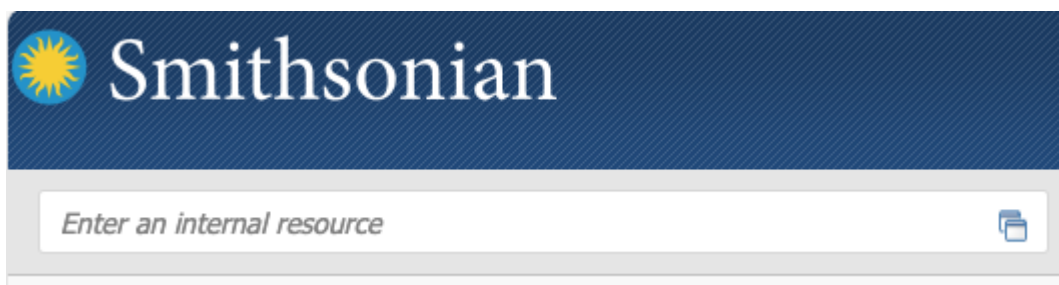
```
$ passwd
Changing password for user USERNAME.
(current) LDAP password:
New password:
Retype new password:
```

The change applies to both login nodes immediately.

Change or reset your password on the self-service page

Use the self-service page to change a password you know, to reset one you have forgotten, or to reset one that expired less than 14 days ago.

1. Open the page. From the SI network or VPN, go directly to <https://hydra.si.edu/ssp/>. From telework.si.edu, choose **Hydra** under IT Tools, then **Change Password on Hydra-7** or **Reset Password on Hydra-7**.
2. To reset, enter your Hydra username and click **Send**. The page emails a link to your canonical address, usually the one ending in `.edu`.
3. Follow the link. From telework, paste it into the **Enter internal resource** box on the telework home page instead of opening it directly, because the link points at a server that only the SI network can reach.



4. Enter your new password.

Unlock an account after 14 days

Once a password has been expired for more than 14 days the account is locked and the self-service page no longer works for it. Email SI-HPC-Admin@si.edu to request a reset. The instructions come to your work email address.

Further reading

- [Quick start](#) for your first job once you are logged in
- [Start an interactive session](#) for a shell on a compute node rather than a login node
- [scp, sftp and rsync](#) which use the same ssh connection to move files

Last updated 2026-10-03

Training and workshops

We require every new Hydra user to watch the introduction video before using the cluster. Beyond that, we run an introductory workshop each quarter and advanced workshops now and then, and their materials stay online.

Introduction to Hydra (required)

Watch the [Introduction to Hydra](#) video. It gives an overview of the cluster, the basics of getting started, and the [usage policies](#) you agree to by using Hydra.

The video is on SI's Microsoft Stream

It opens only from the SI staff network, the SI-STAFF or CfA VPN, or telework.si.edu.

The slides and hands-on material from the introduction are in the [Hydra-introduction](#) repository, so you can follow the exercises at your own pace after the video.

Workshops

The introductory workshop runs every quarter, and we announce it on the HPCC-L mailing list. Advanced workshops run less regularly. Their slides, exercises and recordings are in the [advanced-hydra-workshops](#) repository:

Topic	Date
Job arrays	2023-06-21
Shell scripting and useful tools (sed, awk, grep)	2023-05-10
Installing software and writing modules	2023-03-22
Conda on Hydra	2023-02-15

If you would like a workshop on a topic that is not here, or a session for your group, email SI-HPC@si.edu.

Last updated 2026-09-30

Quick start

A first job on Hydra takes five steps. You make a working directory, write a job file, submit it with `qsub`, check on it with `qstat`, and read the output file at the end. You need an [account](#) and to be [logged in](#). The example job needs no input data and runs for 30 seconds.

Replace `USERNAME` with your Hydra username throughout.

1. Log in and create a working directory

Jobs run from a directory under `/scratch`. The home directory has a small quota and is not for job input and output.

Your directory on `/scratch` is under a group directory such as `/scratch/genomics`, `/scratch/sao` or `/scratch/odi`. Your welcome email tells you which one is yours. The examples on this site use `/scratch/genomics/USERNAME`. Substitute your own path.

```
ssh USERNAME@hydra-login01.si.edu
mkdir -p /scratch/genomics/USERNAME/quickstart
cd /scratch/genomics/USERNAME/quickstart
```

2. Write a job file

A **job file** is a shell script with extra lines at the top. Lines beginning with `#$` are options to the scheduler, and the rest is what the job runs. Create `hello.job` with a text editor (`nano hello.job` if you have no preference):

hello.job

```
#!/bin/sh
#$ -S /bin/sh
#$ -l mres=2G,h_data=2G,h_vmem=2G
#$ -cwd
#$ -j y
#$ -N hello
#$ -o hello.log
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
sleep 30
echo = `date` job $JOB_NAME done
```

Option	Effect
<code>-S /bin/sh</code>	Shell used to run the script
<code>-l mres=2G,h_data=2G,h_vmem=2G</code>	Reserve 2 GB of memory
<code>-cwd</code>	Run in the directory the job was submitted from
<code>-j y</code>	Write error output to the same file as standard output
<code>-N hello</code>	Job name
<code>-o hello.log</code>	Output file

There is no `-q` line, so the job goes to the default queue, `sThC.q`, which allows up to 7 hours of CPU time and is right for most first jobs. [Job script reference](#) lists every option, and the [QSub Generator](#) writes a job file for you from a web form.

3. Submit the job

```
$ qsub hello.job
Your job 15497588 ("hello") has been submitted
```

The number is the **job ID**. `qstat`, `qdel` and `qacct` refer to jobs by this ID, and so will we if you write to us about a job.

4. Check the job

```
$ qstat
job-ID      prior    name         user          state submit/start at
queue                               jclass                      slots ja-task-ID
-----
15497588 0.50500 hello      USERNAME      r      09/30/2026 18:48:14
sThC.q@compute-93-02.cm.cluste                      1
```

State `qw` means queued and waiting, `r` means running. A job with an error in its job file usually fails within seconds, so check `qstat` during the first minute after submitting. When the job has finished, `qstat` prints nothing.

You can log out with `exit` while a job is running. The scheduler owns the job, so it keeps running after you log out.

5. Read the output

```
$ cat hello.log
+ Wed Sep 30 18:48:14 EDT 2026 job hello started in sThC.q with jobID=15497588 on
compute-93-02
= Wed Sep 30 18:48:44 EDT 2026 job hello done
```

`/scratch` **is not permanent storage**

The scrubber deletes files older than 180 days. Move results off the cluster when an analysis is complete. See [Storage](#).

Further reading

- [Data transfer](#) for copying data to Hydra and results back
- [Request a queue, memory and CPUs](#) for a production job that needs more than the default
- [Find and load software](#) for the programs installed on the cluster

Last updated 2026-10-03

Running Jobs

Hydra runs Grid Engine. You submit work as jobs from a login node with `qsub`, and the scheduler runs each job on the compute nodes and kills it if it exceeds the limits of its queue.

Start with the page that fits your job:

- For a first job, [Write and submit a job](#), then [check on it](#).
- For more than the default of 7 hours of CPU, 8 GB and one CPU, [Request a queue, memory and CPUs](#).
- For many similar runs, [Submit a job array](#).
- For a threaded or MPI program, [Submit a parallel job](#).
- For a form in place of a hand-written job file, the [QSub Generator](#).

Do not compute on the login nodes

The login nodes slow, then kill, a process that computes on them. Run it as a job, or under `qrsh` in an [interactive session](#). [Warning emails](#) gives the thresholds.

When you need to look something up, the reference pages in the sidebar cover the [queues](#), the [job script options](#), the [per-user limits](#), the [monitoring tools](#) and the [warning emails](#). If you want to understand how the pieces fit, or you know Slurm and want the differences, read [How the scheduler works](#). For a shell or a notebook on a compute node, see [Interactive Use](#).

Last updated 2026-10-03

How the scheduler works

Every computation on Hydra runs as a **job**, a shell script that you submit from a login node with `qsub`, together with a request for the memory, CPU time and number of CPUs it needs. Each CPU a job is given is called a **slot**. The scheduler places the job on one or more compute nodes and runs it without a terminal.

The scheduler is Grid Engine (Siemens HPCWorks Grid Engine, formerly Sun, Univa and Altair Grid Engine). Documentation and forum posts under any of those names apply.

The parts of the cluster

The cluster consists of:

- two login nodes, `hydra-login01.si.edu` and `hydra-login02.si.edu`, where you edit files, compile, test briefly, copy data, and submit and monitor jobs
- a head node, `hydra-7.si.edu`, which runs the scheduler. Do not log in to it.
- the compute nodes, where jobs run
- the storage systems, described under [Storage](#).

10 Gb Ethernet and InfiniBand connect all the nodes. The nodes run Rocky Linux 8.10, deployed with Bright Cluster Manager 10.

What happens to a job

1. You submit a job file with `qsub`. The scheduler assigns a job ID and puts the job in a queue.
2. The job waits until the resources it requested are free and until you are below your [resource limits](#).
3. The scheduler starts the job on the compute node or nodes it selects. You do not choose the node.
4. The job runs in batch mode. It reads no terminal input, and its standard output and error go to files.
5. If the job exceeds the memory or time limits of its queue, the scheduler kills it.

You check on a job with `qstat` and, after it finishes, with `qacct`. See [Monitor and manage jobs](#).

Types of job

Type	CPUs	How to request	Page
Serial	one	no <code>-pe</code> option	Write and submit a job
Job array	one or more per task	<code>-t</code>	Submit a job array
Multi-threaded	several, all on one node	<code>-pe mthread N</code>	Submit a parallel job
MPI	several, spread over nodes	<code>-pe orte N</code> , <code>-pe ompi N</code> or <code>-pe mpich N</code>	Submit a parallel job
Hybrid	MPI across nodes with threads within each node	<code>-pe hM N</code>	Submit a parallel job

A few compute nodes are set aside for interactive sessions, reached with `qssh`. See [Interactive Use](#).

Queues

Every job runs in a queue, and each queue has limits on CPU time, elapsed time and memory per CPU. **CPU time** is the time a processor is busy with the job, and **elapsed time** is the time on the clock from start to finish. [QSub Generator](#) shows how the two differ.

The queues form a matrix. There is a set of queues each for high-CPU, high-memory and GPU jobs. The high-CPU and high-memory sets have short, medium, long and unlimited time limits, and the GPU set has short, medium and long. Single queues serve extra-large-memory jobs, interactive use, I/O to `/store`, and workflow managers. The scheduler picks a queue from the resources you request, or you can name one with `-q`. If you request the wrong queue or resources, the scheduler rejects the job, leaves it waiting forever, or starts it and kills it. [Queues](#) lists them, and [Request a queue, memory and CPUs](#) explains how to choose.

Limits

Two kinds of limit apply:

- per-queue limits on CPU time, elapsed time and memory, which apply to each job
- cluster-wide limits on how many slots, jobs and how much reserved memory one user can hold at once, which decide when your queued jobs start.

Jobs that would exceed a cluster-wide limit wait in the queue until your other jobs finish. See [Resource limits](#).

Rules for a shared cluster

Do not compute on the login nodes. Use them for editing, compiling, short tests and submitting jobs. The login nodes slow and then kill processes that compute on them.

[Warning emails](#) gives the thresholds. Run anything longer in an [interactive session](#) or as a job.

Start every computation through the scheduler. Do not log in to a compute node and start a program by hand. Use `qsub` or `qrsh`.

Request every CPU your program uses. A program that starts threads or child processes without a matching `-pe` request overloads the node and slows other people's jobs. If your script starts anything in the background, end the script with `wait` so the job does not exit before its processes do. MPI programs are not started the way they are on a workstation. See [Submit a parallel job](#).

Reserve the memory the job uses, and no more. Reserved memory that a job does not use is unavailable to everyone else. See [Reserve memory](#).

Do not submit thousands of very short jobs. Starting a job has overhead. Ten thousand five-minute jobs cost the system as much time to start as they take to run. Group short tasks into fewer, longer jobs. [Job arrays](#) show how.

Give concurrent jobs distinct names and output files. Jobs run at the same time on different nodes, and jobs that write to the same file overwrite each other.

Checkpoint long jobs. Nodes crash, networks fail, and the scheduler kills jobs that exceed a limit. Save intermediate results so a computation can resume from where it stopped. Check whether the software you use supports checkpointing and how to enable it.

Test before you scale up. Run one job, check its CPU and memory use with `qacct`, adjust the request, then submit the rest.

Treat the disks as working space. The scrubber empties the public disks and nothing backs them up. Move results off the cluster when an analysis is complete. See [Storage](#).

Jobs that use far fewer CPUs than requested, more CPUs than requested, or far less memory than reserved trigger [warning emails](#) and can be killed. The [usage policies](#) state the thresholds and what we expect of you.

If you know Slurm

Hydra's commands and options differ from Slurm's. The closest equivalents are:

Slurm	Grid Engine on Hydra
<code>sbatch job.sh</code>	<code>qsub job.sh</code>
<code>#SBATCH</code>	<code>#\$</code>
<code>squeue -u \$USER</code>	<code>qstat -u \$USER</code> or <code>qstat+</code>
<code>scancel JOBID</code>	<code>qdel JOBID</code>
<code>scontrol update</code>	<code>qalter</code>
<code>sacct -j JOBID</code>	<code>qacct -j JOBID</code> or <code>qacct+ -j JOBID</code>
<code>salloc / srun --pty bash</code>	<code>qrsh</code>
<code>--array=1-100</code>	<code>-t 1-100</code>
<code>--cpus-per-task=N</code>	<code>-pe mthread N</code>
<code>--ntasks=N (MPI)</code>	<code>-pe orte N</code> or <code>-pe mpich N</code>
<code>--mem-per-cpu=4G</code>	<code>-l mres=4G,h_data=4G,h_vmem=4G</code> (<code>mres</code> is the job total)
<code>--time=48:00:00</code>	<code>-l s_cpu=48:00:00</code> or <code>-q mThC.q</code>
<code>--dependency=afterany:JOBID</code>	<code>-hold_jid JOBID</code>
<code>\$SLURM_JOB_ID</code>	<code>\$JOB_ID</code>
<code>\$SLURM_ARRAY_TASK_ID</code>	<code>\$SGE_TASK_ID</code>
<code>\$SLURM_NTASKS</code>	<code>\$NSLOTS</code>

Grid Engine limits CPU time as well as elapsed time. For a parallel job the scheduler multiplies the CPU limit by the number of slots and leaves the elapsed limit alone.

Last updated 2026-10-03

Write and submit a job

The simplest job runs a program on one CPU. It takes a job file, `qsub`, and an output file. Every other kind of job starts from the same file with more options. If you need more memory, CPUs or time, see [Request a queue, memory and CPUs](#). If you have many similar runs, see [Submit a job array](#). If your program is threaded or uses MPI, see [Submit a parallel job](#).

Submit a job

1. Change to the directory the job runs in, under `/scratch` or `/data`. The group directory under `/scratch` (`genomics` in the examples) is the one your welcome email names:

```
cd /scratch/genomics/USERNAME/demo
```

2. Create the job file. The `#$` lines are options for `qsub`. The rest is a shell script. `crunch` stands for your own program.

`crunch.job`

```
#$ -S /bin/sh
#$ -N crunch -cwd -j y -o crunch.log
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
./crunch
echo = `date` job $JOB_NAME done
```

3. Submit it:

```
$ qsub crunch.job
Your job 15503721 ("crunch") has been submitted
```

4. Check that it is queued or running:

```
$ qstat
job-ID      prior    name         user          state submit/start at
queue                                  jclass                                                slots ja-task-ID
-----
15503721 0.50500 crunch      USERNAME      r       10/03/2026 16:13:58
sThC.q@compute-76-06.cm.cluste                                           1
```

5. When the job is gone from `qstat`, read its log:

```
$ cat crunch.log
+ Sat Oct 3 16:13:59 EDT 2026 job crunch started in sThC.q with jobID=15503721
on compute-76-06
crunch running with:
result 1.886e+12
= Sat Oct 3 16:14:08 EDT 2026 job crunch done
```

The two `echo` lines record which node and queue the job ran in and when it started and finished. Keep them in every job file. Without `-o` and `-j y`, the output goes to two files, `crunch.oJOBID` and `crunch.eJOBID`. [Job script reference](#) lists every option.

Without `-cwd` the job runs in your home directory

Its output files land there too. Keep `-cwd` in the job file to run in the directory you submit from.

With no queue or resource options the job runs in `sThC.q`, which allows 7 hours of CPU time and 8 GB of memory. Anything larger needs a [queue, memory or CPU request](#).

Pass arguments to the job

One job file serves many runs when the script reads its parameters from the command line. Anything after the file name on the `qsub` line reaches the script as `$1`, `$2`, and so on.

1. Use the arguments in the script:

`crunch.job`

```
## -S /bin/sh
## -N crunch -cwd -j y -o crunch.log
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
OPTIONS="-from $1 -to $2"
echo starting crunch $OPTIONS
./crunch $OPTIONS
echo = `date` job $JOB_NAME done
```

2. Submit each run with its own name and log file, so the runs do not overwrite each other:

```
$ qsub -N crunch-20-50 -o crunch-20-50.log crunch.job 20 50
Your job 15503715 ("crunch-20-50") has been submitted
$ qsub -N crunch-100-150 -o crunch-100-150.log crunch.job 100 150
Your job 15503716 ("crunch-100-150") has been submitted
```

Options on the command line override the `##` lines. For more than a handful of runs, use a [job array](#).

Get email when the job ends

1. Add to the job file:

```
#$ -m abe
```

`b` mails when the job begins, `e` when it ends, `a` when it aborts. Use any subset.

2. Mail goes to the address in your `~/.forward` file on Hydra. To send it elsewhere, add:

```
#$ -M you@example.org
```

Do not request mail for every task of a large job array.

Catch the time limit

Each queue has a soft and a hard time limit 15 minutes apart. At the soft limit the job is sent a signal. At the hard limit the scheduler kills the job. The signal stops the program the job is running, and the job file carries on with its next line. The lines after the program are where you save the state of the run.

Soft limit reached	Signal	The program	A trap in the job file
elapsed time (<code>s_rt</code>)	<code>SIGUSR1</code>	stops with exit status 138, unless it handles the signal itself	runs once the program has stopped
CPU time (<code>s_cpu</code>)	<code>SIGXCPU</code>	stops with <code>CPU time limit exceeded</code> and exit status 152, unless it handles the signal itself	does not run

1. Print the program's exit status after it, and add a `trap` for the elapsed-time signal:

trap.job

```
#$ -S /bin/sh
#$ -cwd -j y -N trap -o trap.log
#
warn()
{
    echo @ `date` warning, received $1 signal.
}
trap "warn usr1" SIGUSR1
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
./crunch
echo "crunch exit status: $?"
echo = `date` job $JOB_NAME done
```

At the soft elapsed-time limit the log shows:

```
User defined signal 1
@ Sun Oct 4 12:46:43 EDT 2026 warning, received usr1 signal.
crunch exit status: 138
```

2. Replace the two `echo` lines with whatever saves the state of the run. The job has until the hard limit, 15 minutes later.

Put checkpointing inside the program where possible, because a program that handles the signal itself is not stopped by it. `cs` scripts cannot catch signals. Use `-S /bin/sh`. [Job script reference](#) lists the signals.

Run jobs in sequence

`-hold_jid JOBID` holds a job until the named job has finished. Use it to chain steps that need different resources, so each step requests only what it uses.

1. Submit the first step:

```
$ qsub -N pre pre-process.job
Your job 15503717 ("pre") has been submitted
```

2. Submit the next step held on the first:

```
$ qsub -hold_jid 15503717 -N main process.job
Your job 15503718 ("main") has been submitted
```

3. `qstat` shows the held job in state `hqw` until the first finishes:

```
$ qstat
job-ID      prior    name          user          state submit/start at
queue                               jclass                      slots ja-task-ID
-----
-----
15503717 0.50500 pre          USERNAME      r      10/03/2026 15:58:58
sThC.q@compute-65-13.cm.cluste                      1
15503718 0.00000 main          USERNAME      hqw    10/03/2026
15:58:57                                              1
```

In a script, capture each job ID with `-terse`, which prints only the ID:

chain.sh

```
#!/bin/sh
parameter=$1
name=$2
jid1=`qsub -terse -N "pre-$name" pre-process.job $parameter`
jid2=`qsub -terse -hold_jid $jid1 -N "process-$name" process.job $parameter`
jid3=`qsub -terse -hold_jid $jid2 -N "post-$name" post-process.job $parameter`
echo submitted $jid1 $jid2 $jid3
```

`qchain` adds the `-hold_jid` options for you. `qchain *.job` submits the matching job files in alphabetical order, each waiting for the previous one. Quote each argument to pass options to `qsub` and arguments to the scripts:

```
qchain '-N start first.job 123' '-N crunch second.job 123' '-N post finish.job 123'
```

Submit many jobs without exceeding the limit

One user may have 2,500 jobs queued at once (see [Resource limits](#)). A script that submits more has to wait for its own jobs to finish.

1. Use `q-wait`, which pauses until jobs whose name contains a string have left the queue, or until at most a given number remain:

```
$ q-wait crunch
# until no job named *crunch* is queued or running
$ q-wait -njobs 125 -wait 3600 crunch # until at most 125 remain, checking
hourly
```

2. Or count with `qstat` in the submission script:

```
NMAX=250
while [ `qstat -u $USER | tail --lines=+3 | wc -l` -ge $NMAX ]
do
    sleep 180
done
```

Add `-s p` to `qstat` to count only pending jobs.

A [job array](#) counts as one job, so an array of 10,000 tasks needs neither.

Further reading

- [Job script reference](#) for every `#$` option, the variables the scheduler sets, and the signals at the time limits
- [Request a queue, memory and CPUs](#) when the default queue is not enough
- [Monitor and manage jobs](#) for what to do once the job is running
- [Examples](#) for complete, tested job files under `~hpc/examples`

Last updated 2026-10-04

Request a queue, memory and CPUs

Four options decide where a job runs and what it may use. They are the queue, the memory reservation, the number of CPUs, and the nodes it may run on. A job that needs more than the default (7 hours of CPU, 8 GB of memory, one CPU, in `sThC.q`) sets one or more of them. The queues and their limits are under [Queues](#).

Choose the queue

1. Estimate the CPU time the job needs per CPU and the memory it needs per CPU. Run one job first if you do not know. `qacct -j JOBID` reports both afterwards (see [Monitor and manage jobs](#)).
2. Add the options for the row that fits, as `#$` lines in the job file or on the `qsub` command line:

Job needs	Add
up to T hours of CPU per CPU	<code>-l s_cpu=T:0:0</code> , or <code>-q</code> with the queue whose class fits
up to T hours of elapsed time	<code>-l s_rt=T:0:0</code>
more than 2 GB per CPU	<code>-l mres=X,h_data=X,h_vmem=X</code> (see Reserve memory)
more than 8 GB per CPU	the same, plus <code>-l himem</code>
more than 450 GB per CPU	<code>-q uTxLM.rq -l himem</code> , restricted queue
a GPU	<code>-l gpu,ngpus=N</code> (see GPUs)
an unknown or unlimited amount of CPU time	<code>-q uThC.q -l lopri</code>
more than one CPU	<code>-pe PE N</code> (see Submit a parallel job)
to read or write <code>/store</code>	<code>-q lTI0.sq -l ioq</code> (see Use /store and the I/O queue)
to submit jobs itself	<code>-q lTWFM.sq -l wfmq</code>

3. Check the request before submitting (see [Check a request](#)).

The scheduler requires `himem`, `gpu` and `lopri` so that it does not place an ordinary job in a high-memory, GPU or unlimited queue only because that queue is less busy. The more a job requests, the fewer similar jobs you can run at once under the [resource limits](#).

Some complete requests are:

Options	Effect
<code>-l s_cpu=48:00:00</code>	48 hours of CPU per CPU; the job lands in a medium queue
<code>-l s_rt=200:00:00</code>	200 hours of elapsed time; the job lands in a long queue
<code>-q mThC.q</code>	run in <code>mThC.q</code>
<code>-l mres=60G,h_data=6G,h_vmem=6G -pe mthread 10</code>	10 CPUs with 6 GB each, 60 GB in total, in a high-CPU queue
<code>-q mThM.q -l mres=12G,h_data=12G,h_vmem=12G,himem</code>	12 GB in the medium high-memory queue
<code>-q uThC.q -l lopri</code>	the unlimited high-CPU queue
<code>-q uTxLM.rq -l himem</code>	the unlimited extra-large-memory queue, restricted

Access to a restricted queue is by request to SI-HPC@si.edu.

Reserve memory

The default queue allows a job 8 GB per slot, and the scheduler kills a job that uses more. A job that uses up to 2 GB per slot needs no reservation. A job that uses more than that reserves it with `mres`. The scheduler tracks reserved memory on every node and does not start a job on a node with less free, unreserved memory than the request. `h_data` and `h_vmem` are the limits at which the scheduler kills the job. A job without a reservation can fail when the node runs short of memory, or crash the node.

1. Set the three values. `mres` is the job total, and `h_data` and `h_vmem` are per CPU. For a serial job the three are equal. For a parallel job, divide the total by the number of slots:

```
#$ -l mres=8G,h_data=8G,h_vmem=8G
```

```
#$ -pe mthread 4
#$ -l mres=32G,h_data=8G,h_vmem=8G
```

2. Add `-l himem` when any per-CPU value is above 8 GB.
3. After the job finishes, compare `maxvmem` from `qacct -j JOBID` with the reservation and lower it if the job used far less.

A memory value without a unit is in bytes

`-l h_vmem=5` limits the job to 5 bytes and it dies at once. Write 5G.

MPI jobs set `h_data` and `h_vmem` only, without `mres`. Reserved memory that a job does not use is unavailable to everyone else, including your own other jobs, and a job that reserves

more than 2.5 times what it uses triggers a [warning email](#). Break a task into separate jobs when its steps need different resources. See [Run jobs in sequence](#).

Restrict the job to certain nodes

1. To run only on nodes in a host group, append `@@GROUP` to the queue name:

```
#$ -q mThC.q@@ib-hosts
```

The queue name can be a pattern. `-q '?ThC.q@@ib-hosts'` means any high-CPU queue on nodes with InfiniBand. [Queues](#) lists the groups. `qconf -shgrp @gpu-hosts` prints the nodes in one.

2. To run only on one CPU architecture, request `cpu_arch`:

```
qsub -l cpu_arch=skylake job.sh          # only skylake nodes
qsub -l cpu_arch='!zen' job.sh           # any node except AMD
qsub -l cpu_arch='haswell|skylake' job.sh # either architecture
```

[Queues](#) lists the architectures.

3. To run one binary per architecture, read the architecture inside the job:

demo2.job

```
#$ -S /bin/sh
#$ -cwd -j y -N demo2 -o demo2.log
#$ -l cpu_arch='!zen'
#
module load tools/cpu_arch
bin/${cpu_arch}/crunch
```

`module load tools/cpu_arch` sets `$cpu_arch` to the node's architecture. `qstat -F cpu_arch -q sThC.q` lists the architectures of the nodes in a queue.

Check a request before submitting

1. Test the job file without submitting it:

```
$ qsub -w v crunch.job
verification: found suitable queue(s)
```

`-w v` checks against an empty cluster. `-w p` checks against the cluster as it is now. `qsub -verify crunch.job` prints what `qstat -j` would show for the job, including the effect of default files and the environment.

2. Submit only when the check passes.

`qsub` applies `-w e` by default, which rejects a request the cluster can never satisfy with:

```
Unable to run job: error: no suitable queues.
```

Do not override this with `-w w` or `-w n`, because the scheduler then accepts the job and it waits forever.

If the job is rejected

The scheduler rejects a job when its request is inconsistent (more CPU time or memory than the queue allows), impossible (more CPUs or memory on one node than any node has), or over a [resource limit](#) (more slots than one user may hold in that queue). `qsub -w v` or `qsub -verify` on the job file says which.

If the job waits and does not start

A job stays in state `qw` when the resources it requested are not free or when you are at a resource limit.

1. Check your usage against the limits:

```
$ qquota -u $USER
```

2. Ask the scheduler why the job has not started:

```
$ qstat -j JOBJD
```

The reason is in the `scheduling info` lines at the end.

A job that waits for hours or days while you are under the limits has requested a scarce resource. Email SI-HPC@si.edu with the job ID.

Further reading

- [Queues](#) for the limits of every queue
- [Resource limits](#) for how much of the cluster one user may hold at once
- [Cluster hardware](#) for the nodes behind each queue and their memory
- [Warning emails](#) for what happens when a job uses far less or far more than it asked for

Last updated 2026-10-03

Submit a job array

A job array runs one job file many times, each run with a different task ID. For more than a handful of similar runs it replaces a loop of `qsub` commands. You submit one job file once, get one job ID, and the scheduler starts the tasks as slots become free.

Submit an array

1. Write a job file that reads its task ID from `$SGE_TASK_ID`. `$TASK_ID` in a `#$` line gives each task its own log file:

model.job

```
## -S /bin/sh
## -N model-1k -cwd -j y -o model.$TASK_ID.log
## -t 1-1000 -tc 100
#
echo + `date` $JOB_NAME started on $HOSTNAME in $QUEUE with jobID=$JOB_ID and ta
skID=$SGE_TASK_ID
#
./model -id model.$SGE_TASK_ID
#
echo = `date` $JOB_NAME for taskID=$SGE_TASK_ID done.
```

`model` stands for your own program. `-t 1-1000` runs tasks 1 to 1000. `-tc 100` runs at most 100 of them at once.

2. Submit it:

```
$ qsub model.job
Your job-array 15503720.1-1000:1 ("model-1k") has been submitted
```

3. Watch the tasks:

```
$ qstat -g d
job-ID      prior    name         user          state submit/start at
queue                                  jclass                                slots ja-task-ID
-----
15503720 0.50500 model-1k    USERNAME      r      10/03/2026 16:00:21
sThC.q@compute-75-03.cm.cluste          1 5
15503720 0.50500 model-1k    USERNAME      r      10/03/2026 16:00:22
sThC.q@compute-75-03.cm.cluste          1 15
15503720 0.50500 model-1k    USERNAME      r      10/03/2026 16:00:22
sThC.q@compute-76-10.cm.cluste          1 17
15503720 0.50500 model-1k    USERNAME      r      10/03/2026 16:00:22
sThC.q@compute-75-03.cm.cluste          1 24
...
15503720 0.00000 model-1k    USERNAME      qw     10/03/2026
16:00:20                                1 999
15503720 0.00000 model-1k    USERNAME      qw     10/03/2026
16:00:20                                1
1000
```

4. Read a task's log, `model.123.log`, once it is done. `qacct -j 15503720 -t 123` reports on one task.

-t value	Tasks
-t 1-20	20 tasks, IDs 1 to 20
-t 10-30	21 tasks, IDs 10 to 30
-t 50-140:10	10 tasks, IDs 50, 60, ..., 140
-t 20	one task, ID 20

An array has at most 10,000 tasks and counts as one job against the 2,500-job limit (see [Resource limits](#)). `-t` combines with `-pe`, and each task then starts as a [parallel job](#) with the requested slots. Do not add `-m abe` to a large array.

Turn the task ID into parameters

Most programs need more than an integer. Three ways to map the task ID to a run, each one line in the job file:

1. One input file per task, `input.1` to `input.1000`:

```
./domodel < input.$SGE_TASK_ID
```

For zero-padded names such as `input.001`, format the ID first:

```
I=`echo $SGE_TASK_ID | awk '{printf "%3.3d", $1}'`
./domodel < input.$I
```

2. One line per task in a parameter file, `parameters-list.txt`:

```
P=`awk "NR==$SGE_TASK_ID" parameters-list.txt`  
./compute $P
```

3. A script of your own, `mytool`, that prints the parameters for a given ID:

```
P=`./mytool $SGE_TASK_ID`  
./compute $P
```

Group short tasks into fewer jobs

Starting a task costs the scheduler time. An array of 5,000 three-minute tasks spends a quarter to a half of its time starting and tracking tasks. A step size gives each task a block of IDs to loop over.

1. Set `-t` with a step, and compute the block in the job file:

`domodel.job`

```
## -S /bin/sh  
## -N model-1k20 -cwd -j y -o model-$TASK_ID-by-20.log  
## -t 1-1000:20  
#  
echo + `date` $JOB_NAME started on $HOSTNAME in $QUEUE with jobID=$JOB_ID  
#  
iFr=$SGE_TASK_ID  
iTo=$(( iFr + SGE_TASK_STEPSIZE - 1 ))  
if [ $iTo -gt $SGE_TASK_LAST ]; then iTo=$SGE_TASK_LAST; fi  
#  
echo running model.sh for taskIDs $iFr to $iTo  
i=$iFr  
while [ $i -le $iTo ]  
do  
    ./model.sh $i > model-$i.log 2>&1  
    i=$(( i + 1 ))  
done  
#  
echo = `date` $JOB_NAME for taskIDs $iFr to $iTo done.
```

2. Put one run in a script the loop calls:

`model.sh`

```
#!/bin/sh  
TID=$1  
echo + `date` model.sh started for taskID=$TID  
./model -id $TID  
echo = `date` model.sh for taskID=$TID done.
```

```
chmod +x model.sh
```

3. Submit `domodel.job`. `-t 1-1000:20` starts 50 tasks with IDs 1, 21, 41, ..., 981, each running 20 models. The array finishes 1,000 models as 50 one-hour tasks instead of 1,000 three-minute ones.

Choose the step to match the run time of one model. A step of 1 is right when each model runs for hours.

Further reading

- [Job script reference](#) for `-t`, `-tc` and `$SGE_TASK_ID`
- [Resource limits](#) for how many tasks may run at once
- [Monitor and manage jobs](#) for watching and deleting tasks

Last updated 2026-10-03

Submit a parallel job

A **parallel job** uses more than one CPU, as threads on one node, as MPI processes across nodes, or both. The program has to be written for one of those, and its documentation says which. A program that runs on one CPU is parallelized by running many copies as a [job array](#).

A parallel job requests a parallel environment (PE) and a number of slots with `-pe PE N` (`N-M` accepts a range) and reads the number it was given from `$NSLOTS`. Memory limits are per slot, so a job with `N` slots may use `N` times the queue's per-slot limit. Reserve it as described under [Reserve memory](#). Only the high-CPU queues run MPI and hybrid jobs. [Job script reference](#) lists the PEs and MPI modules.

Tell the program how many CPUs it was given

`-pe` reserves slots. It does not make the program use them. Pass `$NSLOTS` to the program's thread option, set `OMP_NUM_THREADS=$NSLOTS`, or run `mpirun -np $NSLOTS`. A job that uses fewer or more CPUs than it requested triggers a [warning email](#) and can be killed.

Submit a multi-threaded job

1. Request `mthread` slots and pass `$NSLOTS` to the program. For a program with a thread option, where `demo` stands for your own program and its module:

demo.job

```
## -S /bin/sh
## -cwd -j y -N demo -o demo.log
## -pe mthread 32
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
echo + NSLOTS = $NSLOTS
#
module load tools/demo
demo -threads $NSLOTS
#
echo = `date` job $JOB_NAME done
```

For an OpenMP program, set the variable instead:

hellomp.job

```

## -S /bin/sh
## -cwd -j y -N hellomp -o hellomp.log
## -pe mthread 32
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
echo + NSLOTS = $NSLOTS
#
module load nvidia
export OMP_NUM_THREADS=$NSLOTS
./hellomp
#
echo = `date` job $JOB_NAME done

```

For a program that reads the thread count from a parameter file, write it in with `sed`:

```
sed "s/MTHREADS/$NSLOTS/" gen-params > all-params.
```

2. Submit, then check the log for the `NSLOTS` line and, once the job has run for a while, its CPU use:

```

$ qsub hellomp.job
$ qstat+ +r%

```

A `cpu%` near 100 means every slot is busy. A value near `100/NSLOTS` means the program is running on one CPU.

`~hpc/examples/openmp` has this example built with the GNU, Intel and NVIDIA compilers.

Submit an MPI job

1. Load the MPI module the program was built with (see [MPI modules](#)) and request the matching PE. That is `orte` for OpenMPI and for Intel MPI, `mpi` for NVIDIA's bundled OpenMPI, and `mpich` for MVAPICH.
2. Start the program with the `mpirun` the module defines and `-np $NSLOTS`. With OpenMPI:

hello-orte.job

```

#$ -S /bin/sh
#$ -cwd -j y -N hello -o hello.log
#$ -pe orte 72
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
echo + NSLOTS = $NSLOTS distributed over:
cat $PE_HOSTFILE
#
module load gcc/13.2/openmpi
mpirun -np $NSLOTS ./hello
#
echo = `date` job $JOB_NAME done

```

With MVAPICH, pass the machine file the scheduler writes to `$TMPDIR/machines`:

hello-mpich.job

```

#$ -S /bin/sh
#$ -cwd -j y -N hello -o hello.log
#$ -pe mpich 72
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
echo using $NSLOTS slots on:
sort $TMPDIR/machines | uniq -c
#
module load nvidia/24/mvapich
mpirun -np $NSLOTS -machinefile $TMPDIR/machines ./hello
#
echo = `date` job $JOB_NAME done

```

With NVIDIA's bundled OpenMPI (`nvidia/24/mpi` and `-pe ompi`), pass the host file the scheduler writes to `$TMPDIR/hostfile`:

```
mpirun -np $NSLOTS -hostfile $TMPDIR/hostfile ./hello
```

3. Submit and read the log. The `cat $PE_HOSTFILE` or `uniq -c` line lists the nodes and the slots on each:

```

+ Sat Oct 3 13:46:40 EDT 2026 job hello started in sThC.q with jobID=15503711 on
compute-65-23
+ NSLOTS = 72 distributed over:
compute-65-23.cm.cluster 55 sThC.q@compute-65-23.cm.cluster <NULL>
compute-65-21.cm.cluster 17 sThC.q@compute-65-21.cm.cluster <NULL>
Loading gcc/13.2/openmpi
  Loading requirement: gcc/13.2.0
Hello world from process 19 of 72
...
= Sat Oct 3 13:46:42 EDT 2026 job hello done

```

Do not call `mpirun` by a full path. The module defines `mpirun` for its own version. A mismatched `mpirun` gives unpredictable results. `~hpc/examples/mpi` holds a hello-world job for every compiler and implementation, described in its `README`.

If the log of an MVAICH job shows

```

[proxy:0:0@compute-N-M] HYDU_create_process (utils/launch/launch.c:74): execvp error
on file CODE (No such file or directory)

```

`mpirun` could not find the executable `CODE`. Check the path, and that the file is on a filesystem the compute nodes mount.

Submit a hybrid job

A hybrid job runs K MPI processes on K nodes, each with M threads, for $N = K \times M$ slots. The program must be written for it, usually MPI between processes and OpenMP within each.

1. Request the hybrid PE for M threads per node, `hM`, with N a multiple of M . `-pe h8 64` is 8 nodes with 8 slots each, and `-pe h12 48` is 4 nodes with 12 each.
2. Source the configuration file the scheduler writes before `mpirun`. It resets `NSLOTS` to K , sets `OMP_NUM_THREADS` to M , and rewrites the host file (`$HOSTFILE`, OpenMPI) and machine file (`$MACHINEFILE`, MVAICH) to one entry per node. The `if` lets the same file run as an ordinary MPI job when no hybrid PE was requested:

```

hybrid.job

```

```

## -S /bin/sh
## -q mThC.q -pe h8 64
## -N hybrid -o hybrid.log -cwd -j y
#
echo $JOB_NAME started `date` on $HOSTNAME in $QUEUE jobID=$JOB_ID
#
if [ -e $TMPDIR/set-hybrid-config ]
then
    . $TMPDIR/set-hybrid-config
fi
#
module load gcc/13.2/openmpi
mpirun -np $NSLOTS -hostfile $HOSTFILE ./hybrid
#
echo `date` $JOB_NAME done.

```

3. Submit and read the log. It begins with:

```

----- hybrid_start 1.0/1 -----
hybrid_start: remember to 'source $TMPDIR/set-hybrid-config' to properly setup
your env
-----
...
PE=h8 NSLOTS=8 OMP_NUM_THREADS=8

```

The last line appears only after the configuration file has been sourced.

`~hpc/examples/hybrid` builds and runs a hello-world hybrid program with the GNU, Intel and NVIDIA compilers, with OpenMPI and MVAPICH.

Further reading

- [Compilers, libraries and MPI](#) for building a program against the cluster's MPI
- [Job script reference](#) for the full list of parallel environments
- [GPUs](#) for jobs that use a GPU
- [Examples](#) for tested OpenMP, MPI and hybrid jobs under `~hpc/examples`

Last updated 2026-10-03

Monitor and manage jobs

Once a job is submitted, you watch it with `qstat`, change it with `qalter`, delete it with `qdel`, and read what it used with `qacct` after it finishes. You can also look at one node or at the whole cluster. If you need the full list of options and output fields, see [Monitoring tools](#).

Check on a job

1. List your jobs:

```
$ qstat
job-ID      prior    name         user          state submit/start at
queue                                  jclass                      slots ja-task-ID
-----
15503722  0.50500  model        USERNAME      r      10/03/2026 16:14:03
sThC.q@compute-93-04.cm.cluste          1 1
15503722  0.50500  model        USERNAME      r      10/03/2026 16:14:03
sThC.q@compute-65-03.cm.cluste          1 2
15503722  0.00000  model        USERNAME      qw     10/03/2026
16:14:02                                  1
3-40:1
```

This is a job array with two tasks running and tasks 3 to 40 waiting. `r` is running, `qw` is waiting, `Eqw` is waiting in error and will not run. The [state table](#) has the rest. `qstat -s r` shows only running jobs, `-s p` only pending, `-g d` one line per array task.

2. For a running job, see its age and how much of its requested CPU it uses:

```
$ qstat+ +r%
```

A `cpu%` near 100 means the job uses every slot it requested. A value near `100/NSLOTS` means it runs on one CPU and ignores the rest. `qstat+ +rr%` adds memory and I/O.

3. For a waiting job, ask why:

```
$ qstat -j JOBID
```

The [scheduling info](#) lines at the end name the resource or quota it is waiting for. `qquota -u $USER` shows your usage against the [resource limits](#).

4. For a job in `Eqw`, get the error, delete the job, fix the cause and resubmit:

```
$ qstat -explain E -j JOBID
$ qdel JOBID
```

The usual cause is a directory or file named in the job that does not exist.

Delete a job

1. Delete one job, or all of yours:

```
$ qdel JOBID
$ qdel -u $USER
```

2. To delete some of them, generate the commands from `qstat`, edit the list, run it:

```
$ qstat -u $USER | grep $USER | grep rax | awk '{print "qdel", $1}' > qdel.sh
$ nano qdel.sh
$ sh qdel.sh
```

`grep rax` keeps the jobs whose name contains `rax`. `-s r` or `-s p` on `qstat` limits the list to running or pending jobs.

Change a waiting job

`qalter` changes most properties of a job in `qw`, and a few of a running job.

```
$ qalter -q mThC.q JOBID          # move to another queue
$ qalter -o run-3.log JOBID       # rename the output file
$ qalter -m abe JOBID             # change email notification
$ qalter -mods l_hard s_cpu 240:: JOBID # change a resource that is already in
the request
$ qalter -adds l_hard s_cpu 240:: JOBID # add a resource to the request
```

`qalter -l` replaces the whole resource request

A job submitted with `-l mres=2G,h_data=2G,h_vmem=2G` and altered with `qalter -l s_cpu=1::` is left with `s_cpu` alone. Its memory request is gone. Use `-mods` and `-adds` as above, which change one resource and keep the rest. `qstat -j JOBID` shows the request under `hard_resource_list`. Allow a few seconds for a change to appear there.

Check a finished job

Read the accounting record after every new kind of job and set the memory and CPU requests of the next jobs from it.

1. Print the record:

```
$ qacct -j JOBID
=====
qname                sThC.q
hostname              compute-76-11.cm.cluster
...
granted_pe            mthread
slots                 4
...
failed                0
exit_status           0
ru_wallclock          9.219
...
cpu                   36.362
mem                   0.552
...
maxvmem               145.473M
```

`qacct -j JOBID -t TASKID` reports one task of an array. `qacct+ -j JOBID` reads the same data from a database, faster for old jobs, with selectable fields.

2. Read `maxvmem` against the memory you reserved, `cpu` against `ru_wallclock × slots`, and `failed` and `exit_status` (both 0 when the job completed). [qacct fields](#) lists them.
3. Adjust the request. A job that used 6.3 GB with 32 GB reserved held more than 25 GB back from everyone else. A job whose `cpu` is a quarter of `ru_wallclock × slots` ran on one CPU of the four requested. In the record above, `cpu` is 36.4 s against 9.2 s × 4 slots, so all four slots were busy.

`qacct -d 3 -o $USER -j > qacct.log` saves every job of the past three days for filtering with `egrep`.

Watch a high-memory job over time

A sampler records every job in the high-memory queues every five minutes.

1. Plot the memory and CPU of a job as a PNG:

```
$ module load gnuplot
$ plot-qmemuse JOBID
```

`plot-qmemuse JOBID.TASKID` plots one array task. `-o FILE` names the output, `-x` plots on screen over X11 instead.

2. View the PNG with `display FILE` over an X11 connection, or copy it to your computer.

`show-qmemuse JOBID` prints the same as text. `plot-qssduse` and `show-qssduse` do the same for local SSD use.

Look at a job's processes on its node

1. Find the node:

```
$ qstat+ -nlist JOBID
```

2. Run `top` there:

```
$ rtop+ -u $USER 64-05
```

`rtop+ -50 64-05` shows 50 lines. `rpstree+ 64-05` shows the process tree.

Check the state of the cluster

1. Slots in use, reserved and free in every queue:

```
$ qstat -g c
```

CLUSTER	QUEUE	CQLOAD	USED	RES	AVAIL	TOTAL	aoACDS	cdsuE
all.q		-nan	0	0	0	0	0	0
lTI0.sq		0.00	0	0	34	34	0	0
lTWFM.sq		0.00	0	0	18	18	0	0
lTb2g.q		0.23	0	0	2	2	0	0
lTgpu.q		0.00	0	0	104	104	0	0
lThC.q		0.16	308	0	4308	4616	0	0
lThM.q		0.16	77	0	4539	4616	0	0
...								

`qstat+ -gc` adds node counts and the percentage in use. `show-qslots` prints only the free slots, `qstat+ -es` lists empty slots, `qstat+ -down` lists nodes that are down, and `check-gpu-use` shows the GPUs.

2. One node, or a set of nodes:

```
$ qhost -h compute-64-02 compute-64-03      # CPUs, load, memory
$ qhost -q -h compute-64-02                  # the queues on the node
$ qhost -j -h compute-64-02                  # the jobs on the node
$ qhost | egrep 'LOAD|e-6[45]'               # header plus the compute-64 and
compute-65 nodes
```

`qhost` takes no patterns. Filter with `egrep`.

The [status page](#) shows the same as graphs, with past usage and disk space.

Further reading

- [Monitoring tools](#) for every option and output field of `qstat`, `qstat+`, `qacct` and `qacct+`
- [Warning emails](#) for the checks that run on your jobs and what the emails mean
- [Cluster status](#) for the live view of the queues and nodes

Last updated 2026-10-03

QSub Generator

The QSub Generator is a web form that writes a job file. You fill in the resources the job needs and the commands it runs. The form produces a `.job` file with the matching embedded directives, which you upload to Hydra and submit with `qsub`.

Open <https://hydra.si.edu/tools/QSubGen/> from the SI network, the SI VPN, or telework.si.edu. Use Chrome or Firefox. Safari is not supported. The question-mark icon next to each field explains it and its format.

Fill in the form

Time and memory

Specify the amount of:

CPU time ?	7:00	or	short	time limit queue
Memory ?	1	GB		
VMem ?	1	GB		

CPU time is the CPU time allowed to the job, per CPU. Type a value or pick one from the menu, which sets the text box. The job is killed when it reaches this limit.

CPU time counts only the time a processor is busy with the job. Elapsed time is the time on the clock from start to finish. A job that waits on files uses less CPU time than elapsed time. A job on several CPUs adds up the time of each, so four CPUs that are each busy for 9 seconds use 36 seconds of CPU time. For a job with several CPUs the scheduler multiplies the CPU limit by the number of CPUs and leaves the elapsed-time limit alone. See [Time classes](#).

Memory is the maximum memory the job uses, per CPU. Some programs come with a memory estimator. The form links to one for RAXML.

VMem is the virtual-memory limit. It is set to the memory value unless you change it.

Parallel environment

Select the type of PE: ?

☒ serial ☐ multi-thread ☐ MPI (orte) ☐ MPI (mpich)

Number of CPUs ? 2

Choose serial (one CPU), multi-thread (several CPUs on one node) or MPI (CPUs spread over nodes), and the number of CPUs. The program's documentation and `module help MODULE` say which kinds of parallelism it supports. See [Parallel jobs](#).

GPUs

Use GPUs: ?

☒ none ☐ any ☐ GV100 ☐ LS40S

Number of GPUs ?

Choose whether the job uses GPUs, which type, and how many. The program must be written to use a GPU. See [GPUs](#).

Shell

Select the shell: ?

☒ csh ☐ sh ☐ bash

Choose `sh`. The job files on this site are written for it, and `csh` scripts cannot catch the signals at the time limits.

Modules

Select which modules to add: ?

Type the start of a program name to see the matching modules, and pick the one to load. A module without a version, such as `bioinformatics/raxml`, loads the default version. See [Modules](#).

Commands

Job specific commands: ?

Type your commands here

Enter the commands the job runs, starting with the executable name and followed by its options and input files. `module help MODULE` on Hydra prints the executables a module provides. For an MPI job, start the line with `mpirun -np $NSLOTS`.

Additional options

Additional options: ?

Job Name ?

Log File Name ?

Err File Name ?

☒ Change to cwd ☒ Join stderr & stdout

☒ Send email notifications

Email ?

Field	Effect
Job Name	Name shown by <code>qstat</code> . No spaces.
Log File Name	File for standard output. Filled in from the job name.
Error File Name	File for standard error when output and error are not joined. Filled in from the job name.
Change to CWD	Run the job, and write the log, in the directory you submit from. Leave this checked.
Join stderr & stdout	Write all output to the log file. Check this.
Send email notifications	Email when the job starts and ends.
Email	Address for the notifications.

Check, save and submit

This page is ready!

Check if OK

Save it

1. Click **Check if OK**. The generated script appears in the grey box, with the total CPU time and memory requested above the **Save it** button.
2. Click **Save it** to download the `.job` file.
3. Copy the file to your working directory under `/scratch` or `/data` on Hydra (see [Data transfer](#)).
4. Log in and submit it with `qsub FILE.job`, as in the [Quick start](#).

To change the file afterwards, edit it in a text editor on Hydra. [Job script reference](#) describes every directive it contains.

Last updated 2026-10-03

Queues

Every job runs in a queue, and each queue limits CPU time, elapsed time and memory per CPU. The scheduler selects a queue from the resources a job requests unless the job names one with `-q`. With no options a job runs in `sThC.q`. The default Grid Engine queue `all.q` exists but has no slots. If you are deciding which queue to use, see [Request a queue, memory and CPUs](#). If you want to know which nodes and how many slots stand behind each queue, see [Cluster hardware](#).

Time classes

The high-CPU and high-memory queues each come in four time classes, named by the first letter of the queue. The GPU queues come in the first three.

Class	Letter	Soft CPU time per slot	Soft elapsed time
short	s	7 h	14 h
medium	m	6 d	12 d
long	l	30 d	60 d
unlimited	u	none	none

The hard limits are 15 minutes longer than the soft ones. At a soft limit the scheduler signals the job. At the hard limit it kills the job. For a parallel job the scheduler multiplies the CPU limit by the number of slots and leaves the elapsed limit alone.

Queue sets

Queues	Memory per slot, resident / virtual	Parallel environments	For	Required option
sThC.q mThC.q lThC.q uThC.q	8 GB / 64 GB	mthread, orte, ompi, mpich, hM	serial or parallel jobs needing less than 8 GB per CPU	-l lopri for uThC.q
sThM.q mThM.q lThM.q uThM.q	450 GB / 900 GB	mthread	jobs needing 8 GB to 450 GB per CPU	-l himem; -l lopri for uThM.q
uTxLM.rq	2 TB / 2 TB	mthread	jobs needing more than 450 GB; approved users only	-l himem
lThMuVM.tq	450 GB / unlimited	mthread	high-memory jobs that a virtual-memory limit breaks; 30 d CPU, 60 d elapsed; approved users only	-q lThMuVM.tq -l himem
sTgpu.q mTgpu.q lTgpu.q	64 GB / 256 GB	mthread	jobs that use a GPU	-l gpu, ngpus=N
qrsh.iq	8 GB / 64 GB	mthread	interactive sessions; 12 h CPU, 48 h elapsed	started with qrsh
qgpu.iq	64 GB / 128 GB	mthread	interactive sessions with a GPU; 12 h CPU, 24 h elapsed	qrsh -l gpu, ngpus=N
lTI0.sq	8 GB / 64 GB	mthread	jobs that read or write /store; 48 h CPU, 72 h elapsed	-q lTI0.sq -l ioq
lTWFM.sq	24 GB / 64 GB	mthread	a workflow manager that submits jobs; 6 d CPU, 30 d elapsed, 2 slots	-q lTWFM.sq -l wfmq

Memory limits are per slot, so a job with `-pe mthread 4` in a high-CPU queue may use 4×8 GB. Only the high-CPU queues run multi-node (MPI and hybrid) jobs. In every other queue a parallel job fits on one node.

[qrsh sessions](#) describes the interactive queues, [Use /store and the I/O queue](#) the I/O queue, and [GPUs](#) the GPU queues. Jobs in `lTWFM.sq` run on the `@wfm-hosts` nodes, which can run `qsub`. One such job per user runs at a time. `lTb2g.q` is the queue for the one licensed [Blast2GO](#) job, with 36 GB of memory, one slot, no parallel environment and the long time limits.

Host groups

A host group is a named list of nodes. `-q QUEUE@GROUP` restricts a job to the group. See [Restrict the job to certain nodes](#). `qconf -shgrp1` lists the groups. `qconf -shgrp @GROUP` lists a group's nodes.

Host group	Nodes
@all-hosts	all nodes
@hicpu-hosts	high-CPU nodes
@himem-hosts	high-memory nodes, 512 GB per node
@xlmem-hosts	extra-large-memory nodes, 1 TB or more per node
@io-hosts	nodes in the I/O queue
@wfm-hosts	nodes in the workflow-manager queue
@gpu-hosts	nodes with GPUs
@qrsh-hosts	nodes in the interactive queue
@himemx-hosts	the largest-memory nodes
@b2g-hosts	the Blast2GO node
@ssd-hosts	nodes with a local SSD
@ib-hosts	nodes with InfiniBand
@40c-hosts, @NNc-hosts	nodes with NN CPUs, from 32 to 192
@avx-hosts, @avx2-hosts	nodes whose CPUs support AVX or AVX2

CPU architectures

`-l cpu_arch=NAME` restricts a job to one architecture. `!NAME` excludes one. `A|B` allows either.

Nodes	cpu_arch	CPU
compute-50-*	icelake	Intel Xeon Gold 6338, 2.00 GHz (four L40S GPUs)
compute-64-*	skylake	Intel Xeon Gold 6148, 2.40 GHz; Gold 5218, 2.30 GHz on -17 and -18
compute-65-*	zen	AMD EPYC 7713P 64-core, 1.94 GHz
compute-75-*	zen	AMD EPYC 7H12 64-core, 2.53 GHz
compute-76-*	zen	AMD EPYC 9654 96-core and 9534 64-core
compute-79-*	skylake	Intel Xeon Silver 4114, 2.20 GHz
compute-84-*	skylake	Intel Xeon Platinum 8280, 2.70 GHz
compute-93-*	haswell, broadwell	Intel Xeon E7-8867 v3, 2.50 GHz, and E7-4850 v3, 2.20 GHz (haswell); E7-8860 v4 and E7-8890 v4, 2.20 GHz (broadwell)

Resource value formats

Resource	Format	Examples
memory	number and unit	13.4G, 100M
CPU or elapsed time	hours:minutes:seconds	100:00:00 or 100:: is 100 hours; 100 is 100 seconds

A memory value with no unit is in bytes. `man sge_types` documents the formats.

Query the queue configuration

```
$ qconf -sql # queue names
$ qconf -sq sThC.q # one queue's configuration and limits
$ qconf -sq '?ThM.q' | egrep 'qname|s_cpu|s_rt'
$ qconf -spl # parallel environments
$ qconf -sp mthread # one PE
$ qconf -shgrp1 # host groups
$ qconf -sc | egrep 'cpu_arch|gpu|hmem|lopri|mres' # complexes (the -l resources)
```

`man 5 queue_conf` and `man 5 sge_pe` explain the output.

Last updated 2026-10-04

Job script reference

A job file is a shell script whose `#$` lines carry options for `qsub`. When you need to look up an option, an environment variable, a parallel environment or an MPI module, this is the page. If you are writing your first job file, start with [Write and submit a job](#) instead, which shows how the pieces go together.

Options

Option	Effect
<code>-S /bin/sh</code>	Run the script with the Bourne shell. The default is <code>csh</code> .
<code>-N NAME</code>	Job name, shown by <code>qstat</code> ; sets <code>\$JOB_NAME</code> . No spaces.
<code>-cwd</code>	Run in the directory the job was submitted from and write output files there.
<code>-j y</code>	Write standard error to the standard output file.
<code>-o FILE</code>	Standard output file.
<code>-e FILE</code>	Standard error file, when not using <code>-j y</code> .
<code>-q QUEUE</code>	Run in a named queue ; <code>QUEUE@@GROUP</code> restricts to a host group.
<code>-l RESOURCE=VALUE,...</code>	Request resources: <code>s_cpu</code> , <code>s_rt</code> , <code>mres</code> , <code>h_data</code> , <code>h_vmem</code> , <code>himem</code> , <code>lopri</code> , <code>gpu</code> , <code>ngpus</code> , <code>gpuarch</code> , <code>cpu_arch</code> , <code>ioq</code> , <code>wfmq</code> , <code>ssd_res</code> , <code>idlrt</code> , <code>b2g</code> .
<code>-pe PE N</code>	Request <code>N</code> slots in a parallel environment ; <code>N-M</code> accepts a range.
<code>-t N-M[:S]</code>	Run as a job array with task IDs <code>N</code> to <code>M</code> , step <code>S</code> .
<code>-tc N</code>	Run at most <code>N</code> tasks of an array at once.
<code>-m abe</code>	Email when the job begins, ends, aborts; any subset of the letters.
<code>-M ADDRESS</code>	Email recipient; the default is the address in <code>~/ .forward</code> .
<code>-hold_jid JOBID</code>	Start only after another job has finished.
<code>-terse</code>	Print only the job ID on submission.
<code>-w v</code>	Verify the request against an empty cluster instead of submitting; <code>-w p</code> verifies against the current state.
<code>-verify</code>	Print what <code>qstat -j</code> would show for the job instead of submitting.

`man qsub` lists every option. Options on the command line override `#$` lines.

Where options come from

`qsub` collects options in this order. Each step overrides the previous one.

1. the system-wide file `$SGE_ROOT/$SGE_CELL/common/sge_request`
2. `~/ .sge_request`
3. `.sge_request` in the current directory

4. the `#$` lines in the job file
5. the `qsub` command line

`~/ .sge_request` applies options to every job you submit, one or more per line:

`~/ .sge_request`

```
-cwd -j y
```

Shell

Default shell	<code>csch</code>
Bourne shell	<code>-S /bin/sh</code>
<code>#!</code> line	ignored; the first comment line in the examples is a reminder only
<code>/bin/bash</code>	works; reads startup files that <code>/bin/sh</code> does not, which can change the job's environment
<code>csch</code> limits	cannot catch signals; does not run a last line that lacks a newline

Environment variables

Set by the scheduler in every job. `man qsub` lists the full set.

Variable	Meaning	Example
<code>JOB_NAME</code>	job name from <code>-N</code>	<code>crunch</code>
<code>JOB_ID</code>	job ID	<code>15503724</code>
<code>HOSTNAME</code>	node the job runs on; the master node of a parallel job	<code>compute-64-11</code>
<code>QUEUE</code>	queue the job runs in	<code>sThC.q</code>
<code>NSLOTS</code>	slots allocated by <code>-pe</code>	<code>1</code>
<code>TMPDIR</code>	job-specific temporary directory, deleted when the job ends	<code>/tmp/15503724.1.sThC.q</code>
<code>PE_HOSTFILE</code>	file listing the nodes and slots of a parallel job	
<code>SGE_TASK_ID</code>	task ID in a job array	<code>17</code>
<code>SGE_TASK_FIRST</code> , <code>SGE_TASK_LAST</code> , <code>SGE_TASK_STEPSIZE</code>	the <code>-t</code> range of a job array	<code>1, 1000, 20</code>

In a `#$` line, and only there, `$TASK_ID` expands to the task ID.

Signals at the time limits

Limit	Signal at the soft limit	At the hard limit, 15 min later
CPU time (<code>s_cpu</code> , <code>h_cpu</code>)	<code>SIGXCPU</code>	job killed
elapsed time (<code>s_rt</code> , <code>h_rt</code>)	<code>SIGUSR1</code>	job killed

Parallel environments

PE	Slots are	Queues	Used with
<code>mthread</code>	all on one node	all	threads, OpenMP, any <code>-threads N</code> option
<code>orte</code>	spread across nodes	high-CPU	OpenMPI, Intel MPI
<code>ompi</code>	spread across nodes	high-CPU	NVIDIA's bundled OpenMPI
<code>mpich</code>	spread across nodes	high-CPU	MVAPICH
<code>h2 h4 h8 h12 h16 h24</code> <code>h32 h48 h64</code>	M per node on N/M nodes	high-CPU	hybrid MPI with M threads per process

MPI modules

Every MPI implementation has a build for each compiler. Load the module that matches the compiler the program was built with and the implementation it was linked against.

GCC	Intel	NVIDIA	Provides
	<code>intel/24/mpi</code> , <code>intel/23/mpi</code>	<code>nvidia/24/mpi</code> , <code>nvidia/23/mpi</code>	the vendor's MPI
<code>gcc/13.2/openmpi</code>	<code>intel/24/openmpi</code>	<code>nvidia/24/openmpi</code>	OpenMPI, default version
<code>gcc/13.2/openmpi5</code>	<code>intel/24/openmpi5</code>	<code>nvidia/24/openmpi5</code>	OpenMPI 5
<code>gcc/13.2/openmpi4</code>	<code>intel/24/openmpi4</code>	<code>nvidia/24/openmpi4</code>	OpenMPI 4
<code>gcc/13.2/openmpi4.1.6-13.2.0</code>	<code>intel/24/openmpi4.1.6-24.0</code>	<code>nvidia/24/openmpi4.1.6-24.3</code>	one specific OpenMPI build
<code>gcc/13.2/mvapich</code>	<code>intel/24/mvapich</code>	<code>nvidia/24/mvapich</code>	MVAPICH

```
$ ( module -t avail ) 2>&1 | egrep '^gcc/' | grep mpi
# every MPI module for GCC; intel/, nvidia/ likewise
$ module show gcc/13.2/openmpi                # what the module sets
$ module load gcc/13.2/openmpi; ompi_info      # details of an OpenMPI
build
$ module load gcc/13.2/mvapich; mpirun -info   # details of an MVAPICH
build
```

Every MPI module sets `MPILIB`, `MPIINC` and `MPIBIN`, sets one of `OPENMPI`, `MPICH` or `MVAPICH`, and defines `mpirun` as a shell function or alias for the matching version. `declare -f mpirun` (`sh`) or `alias mpirun` (`csh`) shows which one is active. OpenMPI is not OpenMP. OpenMPI passes messages between processes, and OpenMP runs threads in one process.

Do not use `-V`

`-V` copies your entire login environment into the job. The job then depends on whatever modules and variables were set in that shell, and the same job file fails when submitted later or from a different login. Load modules and set variables inside the job script.

Last updated 2026-10-03

Resource limits

Each queue limits what one job may use. These cluster-wide limits cap how much of the cluster one user may hold at once. If a job would push you over one of these limits, it waits in the queue until your other jobs finish. The commands at the end of the page print the current values.

Queued jobs

Limit	Value
jobs queued on the cluster, all users	25,000
jobs queued by one user	2,500
tasks in one job array	10,000

A job array counts as one job. `q-wait` and a counting loop for scripts that submit many jobs are under [Submit many jobs without exceeding the limit](#).

Slots per user

Queue	Slots
all queues together	840
<code>sThC.q</code>	840
<code>mThC.q</code>	640
<code>lThC.q</code>	431
<code>uThC.q</code>	143
<code>sThM.q</code>	840
<code>mThM.q</code>	640
<code>lThM.q</code>	390
<code>uThM.q</code>	73
<code>uTxLM.rq</code>	536
<code>qrsh.iq</code>	64
<code>lTI0.sq</code>	8
<code>lTWFM.sq</code>	2

Concurrent jobs per user

Queue	Jobs
qrsh.iq	12
qgpu.iq	1
lTIO.sq	8
uTxLM.rq	3
lTWFM.sq	1

Reserved memory per user

Queues	Reserved memory (mres)
sThC.q mThC.q lThC.q uThC.q together	10 TB
sThM.q mThM.q lThM.q uThM.q together	9 TB
uTxLM.rq	8 TB

GPUs per user

Queue	GPUs
all GPU queues together	4
sTgpu.q	4
mTgpu.q	3
lTgpu.q	2
qgpu.iq	1

The cluster has 8 GPUs. A group of approved users has higher GPU limits.

Other limits

Resource	Per user
IDL runtime licenses	102

All users together

Resource	Limit
slots, all queues	5,960
slots in the high-CPU queues	5,176
slots in the high-memory queues	4,680
slots in <code>uTxLM.rq</code>	536
slots in the GPU queues	104
reserved memory, high-CPU queues	40 TB
reserved memory, high-memory queues	36 TB
reserved memory, <code>uTxLM.rq</code>	8 TB

Print the current limits

```
$ qconf -srqs                # every resource quota set
$ qconf -srqs max_hC_slots_per_user # one set
$ qconf -sconf global | grep max    # cluster-wide job limits
$ qquota -u $USER                # your usage against the limits
$ qquota -u $USER -l mem_res       # one resource
$ qquota+ +% -l slots -u $USER     # with percentages
$ check-qwait                    # your waiting jobs and the quota holding each
```

`man 5 sge_resource_quota` and `man 5 sge_conf` explain the output.

Last updated 2026-10-03

Monitoring tools

`qstat`, `qdel`, `qalter`, `qacct`, `qhost` and `qconf` are Grid Engine commands. `qstat+`, `qacct+` and the other tools on this page are Hydra's own, in the `tools/local-user` module, which every session loads. When you need an option or want to know what a field in the output means, look it up here. If you want to see the commands used in order, see [Monitor and manage jobs](#).

qstat

Command	Shows
<code>qstat</code>	your jobs
<code>qstat -u '*'</code>	everyone's jobs
<code>qstat -s r, -s p</code>	only running, only pending jobs
<code>qstat -r</code>	requested resources and full job names
<code>qstat -s r -g t</code>	one line per slot of a parallel job, with the master and the nodes
<code>qstat -g d</code>	one line per task of a job array
<code>qstat -g c</code>	slots used, reserved and free in every queue
<code>qstat -j JOBJID</code>	everything about one job, with why it is waiting
<code>qstat -explain E -j JOBJID</code>	why a job is in error state
<code>qstat -F cpu_arch -q sThC.q</code>	a resource's value on every node of a queue

Job states

State	Meaning
<code>qw</code>	waiting for resources, or held by a resource limit
<code>hqw</code>	held by <code>-hold_jid</code> until another job finishes
<code>r</code>	running
<code>t</code>	being transferred to a node; about to start
<code>d</code>	being deleted
<code>Eqw</code>	in error, waiting; the job will not run

qstat+

`qstat+` (also `q+`) reformats `qstat` output, adding age, CPU efficiency, memory and I/O of running jobs, the node list of a parallel job, and cluster summaries. `qstat+ -help` lists the modes, `qstat+ -examples` shows examples, `man qstat+` has the details. Add `-u USERNAME` or `-u all` for other users' jobs.

Command	Shows
<code>qstat+ +a</code>	all your jobs
<code>qstat+ +a%</code>	all your jobs with age and CPU use as a percentage of age
<code>qstat+ +r%</code>	running jobs with age and CPU efficiency
<code>qstat+ +rr%</code>	running jobs with efficiency, memory and I/O
<code>qstat+ +q</code>	queued jobs
<code>qstat+ -j JOBID</code>	a filtered <code>qstat -j</code>
<code>qstat+ -nlist JOBID</code>	the nodes a parallel job runs on
<code>qstat+ -ineff</code>	jobs using less than 33% of their requested CPUs after 1 hour
<code>qstat+ -osub</code>	jobs using more than 133% of their requested CPUs after 1 hour
<code>qstat+ -ores</code>	jobs that reserved more than 2.5 times the memory they use after 1 hour
<code>qstat+ -gc</code>	the state of every queue with node counts
<code>qstat+ -es</code>	empty slots
<code>qstat+ -down</code>	nodes that are down

`cpu%` is CPU time divided by age and by the number of slots, so 100% means every requested slot is busy.

qacct fields

`qacct -j JOBID` prints the accounting record of a finished job, and `-t TASKID` selects one array task. `qacct -d DAYS -o USERNAME [-j]` covers a user's jobs over DAYS days.

Field	Meaning
qname	queue the job ran in
hostname	node, or master node of a parallel job
taskid	task ID of an array task
qsub_time, start_time, end_time	when the job was submitted, started, ended
granted_pe, slots	parallel environment and slots allocated
failed	0 when the job ran and exited normally; any other value is a failure code, which <code>qacct+ -show explain_failed</code> explains
exit_status	exit status of the job script; 0 when it completed
ru_wallclock	elapsed time, seconds
ru_utime, ru_stime	user and system CPU time reported by the OS, seconds
cpu	CPU time as measured by the scheduler, seconds
mem	memory multiplied by time, GB·s; <code>mem/cpu</code> is the mean memory use in GB
io	a measure of the I/O the job performed
maxvmem	peak memory used

qacct+

A loader copies the accounting data into a PostgreSQL database within about a minute of each job finishing. `qacct+` queries the database, with selectable fields and derived values. Some fields are not loaded correctly from the Grid Engine 8.8.1 records. When a value looks wrong, compare it with `qacct -j JOBID`. `qacct+ -help` and `qacct+ -show help` list the options. `man qacct+` has the details.

Command	Shows
<code>qacct+ -j JOBID</code>	one job
<code>qacct+ -j 8683280:8683290</code>	a range of job IDs
<code>qacct+ -o \$USER -b 7</code>	your jobs submitted in the last 7 days
<code>qacct+ -o \$USER -s "4/27/2026 10:00AM" -u "4/27/2026 11:00AM"</code>	your jobs submitted in a time window
<code>qacct+ -j JOBID -show tab+</code>	tabular output with more fields
<code>qacct+ -j JOBID -show +qname,slots,wallclock=@AGE,cpu=%.1f</code>	chosen fields with formats
<code>qacct+ -show fields</code>	every field available
<code>qacct+ -show explain_failed</code>	meanings of the failed codes

The built-in formats are `simple`, `simple+`, `tab`, `tab+`, `gpu`, `gpu+`, `raw`. A custom `-show` takes `+field,...` for keyed output or `%field,...` for tabular, each field with an optional C format or an `@DATE`, `@MEM` or `@AGE` converter.

qhost and qconf

Command	Shows
<code>qhost</code>	every node: CPUs, load, memory
<code>qhost -h NODE ...</code>	named nodes
<code>qhost -q -h NODE</code>	the queues that include a node
<code>qhost -j -h NODE</code>	the jobs on a node
<code>qhost -ncb</code>	the classic column layout
<code>qconf -sql</code> , <code>qconf -sq QUEUE</code>	queue names; one queue's configuration
<code>qconf -spl</code> , <code>qconf -sp PE</code>	parallel environments; one PE
<code>qconf -shgrpl</code> , <code>qconf -shgrp @GROUP</code>	host groups; one group's nodes
<code>qconf -srqs [NAME]</code>	resource quotas
<code>qconf -sconf global \ grep max</code>	cluster-wide job limits

`qhost` takes no patterns. Filter with `egrep`.

Hydra tools

Every session loads `tools/local-user`. `tools/local` adds `tools/local-admin`, and `tools/local+` and `tools/misc` add more. Every tool has a man page, and `module help tools/local-user` lists them.

Tool	Module	Purpose
<code>qstat+</code> , <code>q+</code>	local-user	<code>qstat</code> with efficiency, memory and cluster summaries
<code>qacct+</code>	local-user	accounting from the database
<code>qquota+</code>	local-user	queue quotas against their limits, with <code>+%</code> for percentages
<code>quota+</code>	local-user	disk quotas on every filesystem; see Quotas
<code>check-qwait</code>	local-user	waiting jobs and the quota holding each
<code>check-gpu-use</code> , <code>get-gpu-info</code>	local-user	GPU use across the cluster; GPUs on the current node
<code>qchain</code>	local-user	submit jobs that run in sequence
<code>q-wait</code>	local-user	pause until jobs leave the queue
<code>plot-qmemuse</code> , <code>show-qmemuse</code>	local-user	memory and CPU of a high-memory job over time
<code>plot-qssduse</code> , <code>show-qssduse</code>	local-user	local SSD use of a job over time
<code>rtop+</code> , <code>rpstree+</code>	local-user	<code>top</code> and <code>pstree</code> on a compute node
<code>show-qslots</code>	local-user	free slots per queue
<code>monitor-code-usage</code>	local-user	how much a code of yours runs
<code>rwho</code> , <code>ruptime</code>	local-user	users and load on the head, login and storage nodes
<code>ghost+</code>	local-admin	<code>ghost</code> with more columns
<code>check-memres</code>	local-admin	memory reserved against memory used
<code>check-qlogs</code>	local-admin	the reports behind the warning emails
<code>check-memuse</code> , <code>check-hi-memuse</code>	local-admin	cluster memory and CPU use
<code>get-cpu_arch</code>	local-admin	the current node's CPU architecture
<code>disk-usage</code>		

Tool	Module	Purpose
	local-admin	how full each partition is; see Quotas
plot-qssduse-summary	local-admin	local SSD use across the cluster
elapsed, lsth, lswc, tails, total, fixFmt, p-wait, procinfo, get-jobhr, noX, useX, dus-report, check-qacct	local+	shell utilities
dua, dut, gdu, ncd, dus+	misc	disk-usage tools

Last updated 2026-10-03

Warning emails

Automated checks watch the login nodes, running jobs and the disks, and email you when something you own is outside the limits. The mail goes to the address in your `~/ .forward` file on Hydra. The [usage policies](#) state the thresholds and require you to act on the warnings. We kill jobs that are not corrected.

Subject	Trigger	Check
Process N priority was lowered on hydra-login0X.si.edu	a process on a login node above 45% CPU for 20 min	continuous
Process N was killed on hydra-login0X.si.edu	a login-node process above 85% CPU or 55% memory	continuous
Your inefficient job(s) on Hydra	jobs using under 33% of their requested CPUs	weekly
You have N running job(s) that use almost no CPU cycles: "hosed"	jobs under 10% efficiency after 36 h	daily
Your oversubscribed job(s) on Hydra	jobs using over 133% of their requested CPUs	weekly
Memory Over-Reservation Warning	high-memory jobs older than 12 h that reserved over 2.5× what they use	Tuesday and Friday
Warning: your disk usage is above 95% of your quota	your usage over 95% of a quota	daily
Warning: disk usage check found N disk(s) with %Use or %IUse >= 95%	a disk you have files on is over 95% full	daily

High CPU use on a login node

The login nodes are for editing, compiling, short tests, transfers and submitting jobs. A process that runs on one for more than 20 minutes at more than 45% CPU has its priority lowered (`renice +5`), and this happens twice. A process that reaches 85% CPU or 55% of the node's memory gets `kill -9`.

Subject: Process 1259315 priority was lowered on hydra-login01.si.edu

Your command `'/process/causing/high-CPU'` (PID=1259315 on hydra-login01.si.edu) priority was lowered (`renice +5`) because:

used TIME = 26.6 > 20 min

used %CPU = 98.4 > 45 %

Remember: jobs/long computations should be submitted to a queue, or run in the interactive queue, not on a login node.

Stop the process and run it as a [job](#) or in an [interactive session](#). These programs commonly trigger it:

- Analyses: submit them as jobs or run them under `qcrsh`.
- `conda`: the "solving environment" step uses a full CPU. Run `conda` under `qcrsh`, use `mamba`, and install into a new environment rather than modifying an existing one. See [Python and conda](#).
- `gzip`, `zip`, `tar`: run them under `qcrsh` or as a job.
- File transfers: some transfer programs use a full CPU. Email SI-HPC@si.edu for alternatives.

After a kill, check the process's output. A transfer may have left partial files. A `tar` archive being written is incomplete and corrupt.

Inefficient jobs

A job is inefficient when its CPU use, divided by its age and by the number of slots requested, is under 33%. The weekly mail lists your inefficient jobs of the past seven days, running or finished, with the CPU-days each wasted, and shows the command that produced the report:

```
$ module load tools/local
$ check-qlogs ineff -from -7d -user $USER
```

Column	Meaning
age	how long the job has run, as HH:MM or +DD:HH
nPEs	slots requested
cpu%	the share of those slots in use; 50% means half the requested CPUs were busy on average
unused CPUs	CPU-days requested and not used

A `cpu%` near $100/nPEs$ means the program ran on one CPU because the thread or process count was not passed to it. Check the program's option for the number of threads and set it from `$NSLOTS`, as shown under [Submit a parallel job](#). Programs whose CPU use varies by stage, as in pipelines where only some steps are parallel, also show low efficiency. Split such pipelines into [jobs run in sequence](#) that each request what they use.

Correct the request in the next jobs you submit. A running job does not need to be deleted for this warning alone. When the cluster load is over 70%, users with many inefficient jobs have jobs killed automatically down to 100 unused slots per user.

Hosed jobs

A daily check flags jobs that have run for more than 36 hours at under 10% efficiency:

The following job is running but using almost no CPU cycles i.e.: efficiency (CPU/age) < 10% and age > 36hr

jobID	name	user	age	nPEs	cpu%	queue	node	taskID
1234570	job4	USER	+3:13	10	9.9%	lThM.q	64-17	

Either the program has stalled after doing some work, in which case kill it with `qdel`, or it is running on one CPU of the ten requested. In the example above, `cpu%` is 9.9 against a `100/nPEs` of 10. Reply to SI-HPC-Admin@si.edu within 24 hours to say whether the job should be killed.

Oversubscribed jobs

A job is oversubscribed when it uses more than 133% of the CPUs it requested. It slows every other job on the node. The weekly mail lists such jobs with the excess CPU-days each used, from `check-qlogs osub -from -7d -user $USER`.

The usual cause is a program that uses every CPU on the node unless told otherwise. Pass `$NSLOTS` to its thread option, or request the slots it uses with `-pe mthread N`. Kill an oversubscribed job that will run for more than another 24 hours and resubmit it with the correct request. Reply to SI-HPC-Admin@si.edu within 24 hours. We kill oversubscribed jobs when the cluster is busy.

Memory over-reservation

Twice a week a check compares each job in the high-memory queues with the memory it reserved with `memres`. A job older than 12 hours that has reserved more than 2.5 times its peak use (`maxvmem`) triggers the warning. The mail shows the `check-memres -details -u USERNAME` report with the ratio for each job.

Reserved memory that is not used is unavailable to every other high-memory job. Read `maxvmem` from `qacct -j JOBID` or `qacct+` when the job finishes, since memory use varies during a run, and set the next reservation from it. See [Reserve memory](#).

Disk quota over 95%

The mail lists each filesystem where your space or file count is above 95% of your quota, marked `***` where it is at or above 100%. Writes fail once you reach a quota. Delete or move files, and consolidate large numbers of small files into an archive. See [Quotas](#).

A disk over 95% full

The mail lists a filesystem that is over 95% full, or over 95% of its inodes, and the top users on it. You receive it because you have files there. Delete or move what you no longer need. See [Storage](#).

Examples

`~hpc/examples` on Hydra holds small, complete test cases, each with source code, a `Makefile`, a job file and the log the job produced. Copy a directory to your own space, build and submit it, and use it as the starting point for your own job.

```
$ find ~hpc/examples -type d
$ cp -r ~hpc/examples/serial /scratch/genomics/USERNAME/
```

Directory	Contents
<code>serial/</code>	hello-world serial job, one per compiler (GCC, Intel, NVIDIA)
<code>mpi/</code>	MPI jobs for each compiler and implementation (vendor, MVAPICH, OpenMPI 3, 4 and 5); see Submit a parallel job
<code>openmp/</code>	OpenMP jobs for each compiler
<code>hybrid/</code>	MPI plus OpenMP jobs using the hybrid PEs
<code>gpu/</code>	GPU jobs and timings; see GPUs
<code>containers/</code>	a Singularity job; see Containers
<code>python/</code>	the Python installations available and how to run each
<code>idl/</code>	IDL, GDL and FL jobs
<code>java/</code>	a Java job
<code>matlab/</code>	a MATLAB runtime job
<code>lapack/</code>	linking with Intel MKL and LAPACK
<code>gsl/</code>	a program using GSL
<code>c++11/</code>	a C++11 build
<code>memtest/</code>	large-memory use and reservation
<code>ssd/</code>	using a node's local SSD; see Use a node's local SSD
<code>misc/</code>	other examples

Each directory has a `README`. The [Quick start](#) walks through a first job. The guides under [Software](#) show job files for specific packages.

Last updated 2026-09-30

Cluster hardware

Hydra has 74 compute nodes, two login nodes and three storage systems, connected by Ethernet and InfiniBand. If you want to know which nodes a queue runs on or what a node has, the tables on this page answer that. For the live picture, `qhost` on a login node prints the node list and `qstat -g c` prints the slots per queue.

Head and login nodes

Node	Role
<code>hydra-7.si.edu</code>	head node: runs the scheduler and starts jobs. Do not log in to it.
<code>hydra-login01.si.edu</code> , <code>hydra-login02.si.edu</code>	login nodes: edit, compile, submit and monitor jobs, transfer data. Use either. They are shared by everyone who is logged in and are not for computing.
<code>hydra-globus01</code> , <code>hydra-globus02</code>	Globus endpoint nodes behind the Hydra collections; see Globus .

The login nodes slow, then [kill](#), computations run on them.

Compute nodes

74 nodes, 5,840 CPU cores, 49 TB of memory and 8 GPUs. Nodes are named `compute-NN-MM`. The first number groups a model. Jobs reach them only through the scheduler. See [How the scheduler works](#).

Nodes	Count	Cores per node	Memory per node	CPU	Note
compute-50-01	1	64	512 GB	Intel Xeon Gold 6338 (icelake)	four NVIDIA L40S GPUs
compute-64-03 to -16 , without -05	13	40	384 GB	Intel Xeon Gold 6148 (skylake)	
compute-64-17 , -18	2	32	512 GB	Intel Xeon Gold 5218 (skylake)	
compute-65-02 to -30 , without -08	28	64	512 GB	AMD EPYC 7713P (zen)	
compute-75-03 to -07	5	128	768 GB	AMD EPYC 7H12 (zen)	
compute-75-01 , -02	2	128	1 TB	AMD EPYC 7H12 (zen)	
compute-76-03 to -14	12	128	1 TB	AMD EPYC 9534 (zen)	
compute-76-01 , -02	2	192	1.5 TB	AMD EPYC 9654 (zen)	-02 runs the RStudio server
compute-79-01 , -02	2	20	128 GB	Intel Xeon Silver 4114 (skylake)	two NVIDIA GV100 GPUs each
compute-84-01	1	112	896 GB	Intel Xeon Platinum 8280 (skylake)	
compute-93-01	1	64	512 GB	Intel Xeon E7-8867 v3 (haswell)	
compute-93-02 to -04	3	72	768 GB	Intel Xeon E7-8860 v4 (broadwell)	
compute-93-05	1	96	2 TB	Intel Xeon E7-8890 v4 (broadwell)	extra-large memory
compute-93-06	1	56	3 TB	Intel Xeon E7-4850 v3 (haswell)	extra-large memory

The name in parentheses is the `cpu_arch` value for restricting a job to that architecture, as shown under [Restrict the job to certain nodes](#). A few nodes are set aside for [interactive sessions](#) and the [I/O queue](#). [GPUs](#) describes the cards.

Hardware behind each queue

Slots are from `qstat -g c`. A node serves several queues, so the same slots appear in more than one row.

Queues	Slots	CPUs per node	Memory	Note
sThC.q mThC.q lThC.q uThC.q	4,616 to 4,656	40 to 128	more than 4 GB per CPU	high-CPU queues
sThM.q mThM.q lThM.q uThM.q	4,616 to 5,064	32 to 192	512 GB or more per node	high-memory queues
uTxLM.rq	536	56 to 192	1 TB or more per node	extra-large-memory queue, restricted
sTgpu.q mTgpu.q lTgpu.q	104, with 8 GPUs			GPU queues; -l gpu, ngpus=N
qgpu.iq	104, with 8 GPUs			interactive GPU queue; qrsh -l gpu, ngpus=N
qrsh.iq	292			interactive queue; qrsh
lTIO.sq	34			I/O queue for /store
lTWFM.sq	18			workflow-manager queue
lThMuVM.tq	384			long high-memory queue with no virtual-memory limit, restricted
lTb2g.q	2			Blast2GO queue; one slot in use at a time

The per-job limits of each queue are under [Queues](#) and the per-user limits under [Resource limits](#).

Network

All nodes are on 10 Gb Ethernet. The head, login and compute nodes are also on a 100 Gb InfiniBand director switch, which carries MPI traffic and I/O to /scratch.

Storage systems

System	Partitions	Access
NetApp FAS8300, two controllers, about 575 TB	/home, /data	all nodes, over NFS on Ethernet
GPFS, two NSD servers, about 3.2 PB	/scratch	all nodes, over InfiniBand
NAS, two systems, about 2.6 PB	/store	login, head, interactive and RStudio server nodes only

The partitions, their quotas and retention rules are under [Filesystems](#).

Last updated 2026-10-04

Interactive Use

Interactive work runs on a compute node. You get there with `qrsh`, which gives you a shell on a node for up to 48 hours. Jupyter, VS Code and RStudio run on that node too, and an ssh tunnel shows them in a browser on your own machine. The [RStudio server](#) is the exception. It is a dedicated node with its own web address and needs no tunnel.

Do not run interactive work on the login nodes

The login nodes slow and then kill processes that compute on them, which ends a notebook or an editor session with it. Start a [session on a compute node](#) first. [Warning emails](#) gives the thresholds.

Read [Start an interactive session](#) first. It shows you how to get a session with `qrsh`, what the limits are, and how to open the tunnel the other pages rely on. Then go to the page for the tool you use, whether that is [Jupyter](#), [VS Code](#), the [RStudio server](#) or [RStudio on a compute node](#). If your work does not need a terminal or a browser, submit it as a job instead. See [Running Jobs](#).

Last updated 2026-10-03

Start an interactive session

`qssh` gives you a shell on a compute node, scheduled like a job into the interactive queue `qssh.iq`. Use it for anything that runs longer than a few minutes and needs a terminal, such as compiling, testing a job before submitting it, `conda install`, or a notebook or editor served to your browser.

Start a session

1. On a login node, request a session. With no options you get one CPU and 8 GB:

```
$ qssh
```

For several CPUs on one node, or more memory, request slots. Memory limits multiply with the slot count:

```
$ qssh -pe mthread 4
```

For a GPU, request one in the interactive GPU queue (see [GPUs](#)):

```
$ qssh -l gpu,ngpus=1
```

2. Wait for the prompt to change from the login node to a compute node:

```
[USERNAME@compute-64-15 ~]$
```

The session starts in your home directory. `cd` to your working directory under `/scratch`.

3. Work as you would on a login node. Load modules, run programs, edit files.
4. End the session with `exit`. A session also ends when it reaches its time limit.

For a program that opens X11 windows, use `qlogin` in place of `qssh` and connect to the login node with X forwarding (`ssh -Y`). `qlogin` takes the same options.

Limits

Limit	Value
CPU time per session	12 h per slot
Elapsed time per session	48 h
Memory per slot	8 GB resident, 64 GB virtual
Sessions per user at once	12
Slots per user across sessions	64
GPU sessions per user	1, with 1 GPU, for up to 24 h elapsed

The scheduler kills a session that reaches a limit, with whatever it was running. Save work in files as you go, and run anything longer than 48 hours as a [job](#).

Reach a service on the node from your browser

Jupyter, VS Code and RStudio run on the compute node and listen on a port there. The compute nodes are not reachable from outside the cluster, so you open an ssh tunnel from your own machine through a login node to that port, then point a browser at the local end. The tools' start scripts print the exact command. It always has this shape:

```
$ ssh -N -L PORT:compute-XX-XX:PORT USERNAME@hydra-login01.si.edu
```

`-N` opens no shell, `-L` forwards local `PORT` to `PORT` on the node. The command asks for your Hydra password and then prints nothing while the tunnel is open. Leave that terminal alone. Use the login node you connected to (`hydra-login01` or `hydra-login02`). The first `PORT` can be any free port on your machine if the same number is in use locally. To close the tunnel, press `Ctrl-C` in that terminal.

The tunnel does not work from telework.si.edu

A tunnel runs from your own machine, and the telework web terminal cannot start one. From telework, use the [RStudio server](#) or a [VS Code tunnel](#) instead. Neither needs an ssh tunnel.

Further reading

- [Queues](#) for the interactive queue's limits
- [Resource limits](#) for how many sessions and slots one user may hold
- [Write and submit a job](#) for work that does not need a terminal

Last updated 2026-10-03

Jupyter

Jupyter Lab runs on a compute node, and an ssh tunnel brings it to a browser on your machine. It keeps a record of an analysis in notebook form and shows images and plots that a terminal cannot.

Start a Jupyter Lab server

1. Start an [interactive session](#), with slots and a GPU if the notebook needs them:

```
$ qcrsh -pe mthread 4
```

2. On the compute node, change to the directory the notebooks live in, or above it, because the server serves that tree.
3. Load a Python. Either the Python module:

```
$ module load tools/python
```

or a conda environment that has Jupyter installed (see [Python and conda](#)):

```
$ module load tools/conda
$ start-conda
$ source activate ENVIRONMENT
```

With a GPU, also load the CUDA libraries: `module load nvidia/24/cuda`. `nvidia-smi` then lists the card.

4. Start the server with the script in `tools/jupyter`, which picks a port and prints the tunnel command:

```
$ module load tools/jupyter
$ start-jupyter-lab-server
```

The script prints the tunnel command and waits. Press Enter to start the server, or start the script with `-no-pause` to skip the wait. `-port N` chooses the port. The default is 8888. `-help` lists the options. Without the script, the command is

`jupyter lab --no-browser --ip=$(hostname) --port=8888`, and the output ends with a URL of the form `http://compute-XX-XX:8888/lab?token=...`.

5. On your own machine, open the tunnel the script printed, in a terminal you then leave alone:

```
$ ssh -N -L 8888:compute-XX-XX:8888 USERNAME@hydra-login01.si.edu
```

6. Open <http://localhost:8888> in a browser and paste the token from the server's output when asked, or open `http://localhost:8888/lab?token=TOKEN` directly.

Stop the server

1. In the browser, choose **File** and then **Shut Down**. This stops the notebook and the server on the node.
2. `Ctrl-C` in the tunnel's terminal.
3. `exit` in the interactive session.

Use a password instead of the token

Once, on a compute node with the same Python loaded:

```
$ jupyter server password
```

Later servers ask for that password in place of the token.

Use a conda environment as a kernel

To run notebooks against the packages of a conda environment, register it as a kernel once:

```
$ source activate ENVIRONMENT
$ python -m ipykernel install --user --name=ENVIRONMENT
```

The environment then appears in Jupyter's kernel list. A registered kernel also lets a notebook run as a batch job with [Papermill](#).

Further reading

- [Python and conda](#) for installing packages and kernels in your own environment
- [Start an interactive session](#) for the session and tunnel this page builds on

Last updated 2026-10-03

VS Code

VS Code reaches Hydra three ways. Remote-SSH connects the VS Code on your own machine to a login node, which is fine for editing files and nothing more. The login-node limits apply to anything it runs. The other two run VS Code itself on a compute node and show it in your browser, through an ssh tunnel or through a tunnel that Microsoft relays.

Way	Where VS Code runs	Needs	Use for
Remote-SSH	your machine; a helper on a login node	the Remote-SSH extension	editing files; nothing that computes
VS Code server	a compute node, shown in your browser	an ssh tunnel from your machine	any interactive work
VS Code tunnel	a compute node, shown in your browser	a GitHub account; no ssh tunnel	the same, and from telework.si.edu

The `tools/vscode` module also provides `vscode-desktop`, a windowed VS Code over X11, which needs an X server that supports it and is slow. The module's `start-vscode` script offers desktop, server and tunnel at a prompt.

Use Remote-SSH from your own VS Code

1. Install VS Code and the **Remote-SSH** extension on your machine.
2. **Remote-SSH: Connect to Host**, enter `USERNAME@hydra-login01.si.edu`, and log in with your Hydra password.

Remote-SSH runs its helper on a login node

Anything you start from its terminal is under the login-node limits. For anything that computes, submit a job or open an [interactive session](#).

Start a VS Code server on a compute node

1. On a login node, start an [interactive session](#), then load the module and start the server:

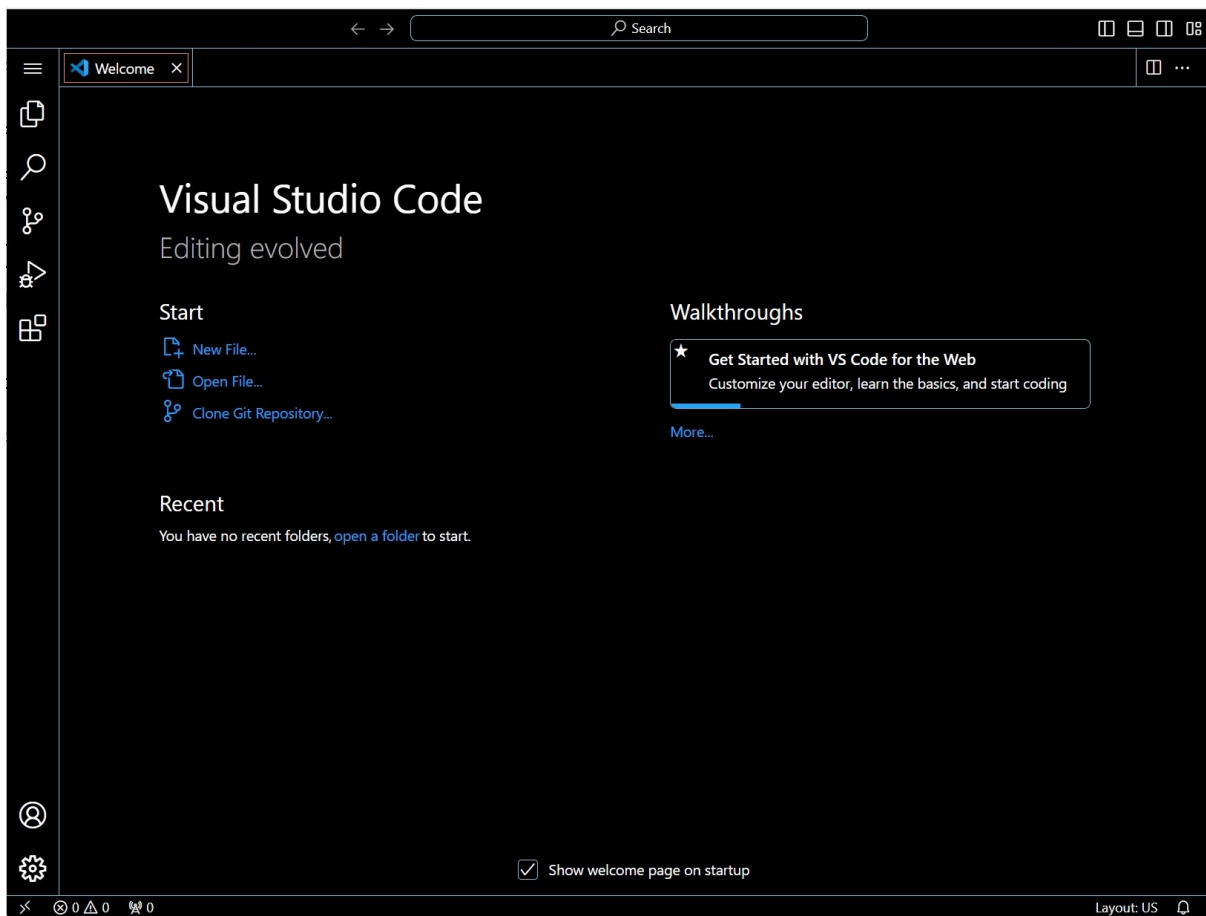
```
$ qrsh
$ module load tools/vscode
$ start-vscode-server
start-vscode-server: starting vscode server on host=compute-64-15
IP=192.168.92.85 and port=8000
    you need to create a ssh tunnel on your local machine with
        ssh -N -L 8000:192.168.92.85:8000 USERNAME@hydra-login01.si.edu
    in a terminal window.
...
Web UI available at http://192.168.92.85:8000?tkn=xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx
```

The script prints the tunnel command and waits. Press Enter to start the server, or start the script with `-no-pause` to skip the wait. The token changes every start. `-token STRING` fixes it and `-no-token` removes it. `start-vscode-server -help` lists the options.

2. On your own machine, open the tunnel the script printed, in a terminal you leave alone:

```
$ ssh -N -L 8000:192.168.92.85:8000 USERNAME@hydra-login01.si.edu
```

3. In a browser, open `http://localhost:8000?tkn=TOKEN` with the token from step 1. Use `localhost`. The `192.168....` address the server prints is reachable only inside the cluster.



4. When done, `Ctrl-C` in the server's window, `Ctrl-C` in the tunnel's terminal, then `exit` the session.

Use a VS Code tunnel

A tunnel registers the compute node with Microsoft's relay under a name, and you open `vscode.dev` with that name. It needs a GitHub account for authentication. SI Microsoft accounts fail because of SI's MFA setup.

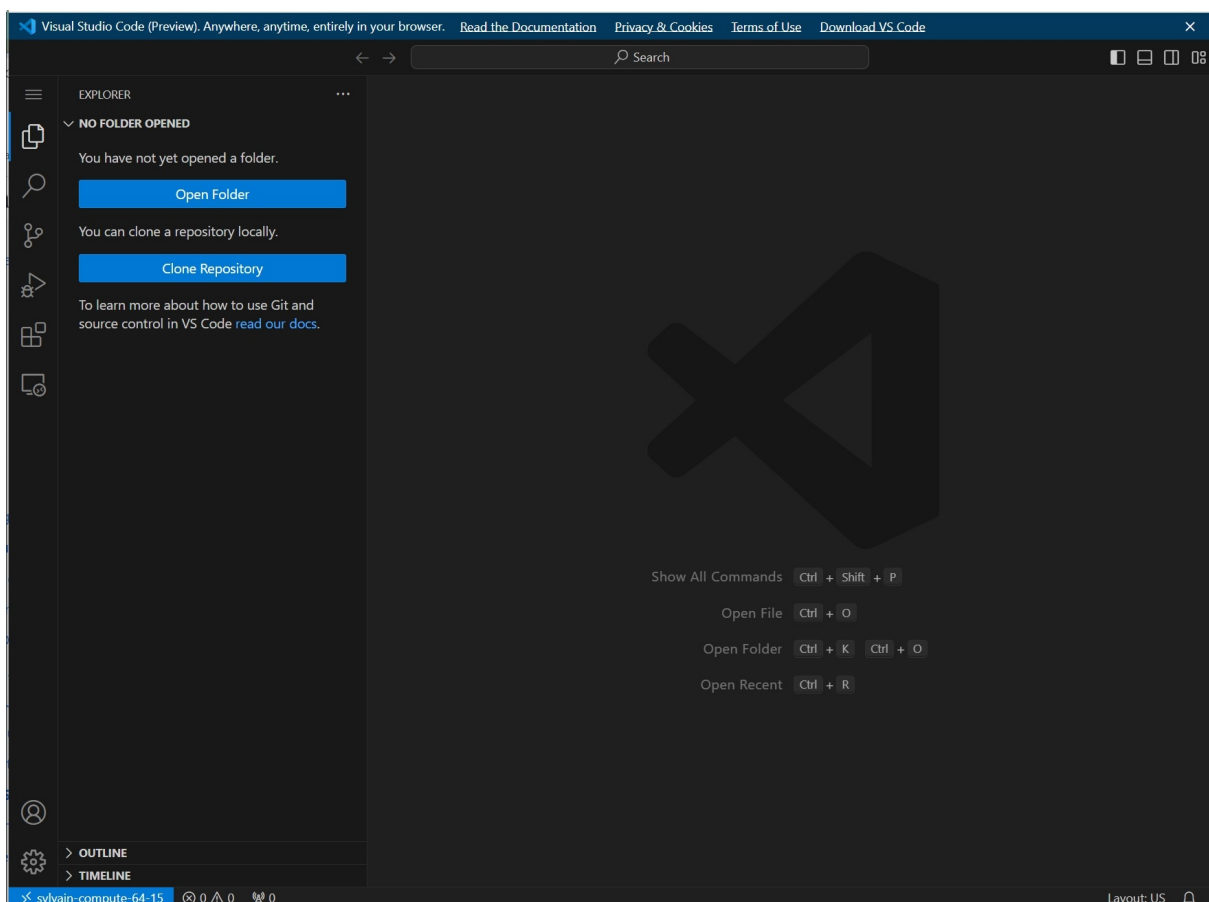
1. In an interactive session, load the module and start the tunnel:

```

$ qrsh
$ module load tools/vscode
$ start-vscode-tunnel
start-vscode-tunnel: starting a tunnel, name: USERNAME-compute-64-15
  to access the tunnel, point your browser to https://vscode.dev/tunnel/
  USERNAME-compute-64-15 on your local machine.
...
? How would you like to log in to Visual Studio Code? >
  Microsoft Account
[] GitHub Account
To grant access to the server, please log into https://github.com/login/device
and use code ABE4-F1A6

```

2. Press Enter when the script asks, to start the tunnel. Then choose **GitHub Account** with the arrow keys, open the device-login URL in a browser, and enter the code.
3. Open `https://vscode.dev/tunnel/USERNAME-compute-64-15` in a browser. Choose **GitHub** if it asks how to authenticate.



4. When done, `Ctrl-C` in the tunnel's window, then `exit` the session.

If the tunnel fails at start with `error failed to lookup tunnel: connection error`, run `host use.rel.tunnels.api.visualstudio.com` once, which returns an address, and start the tunnel again. To reset the stored credentials, delete `~/.vscode/cli/token.json`.

Further reading

- [Start an interactive session](#) for the session and tunnel this page builds on
- [Find and load software](#) for the compilers and interpreters VS Code will find on the node

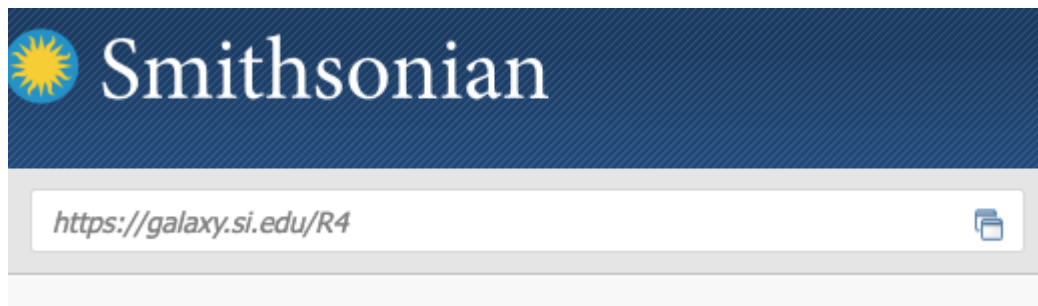
Last updated 2026-10-03

RStudio server

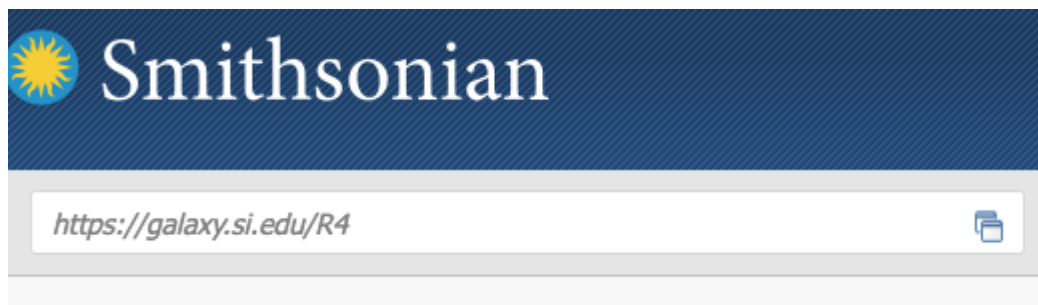
Hydra has a dedicated RStudio server at <https://galaxy.si.edu/R4>. A browser opens an RStudio session on a node with 192 CPU cores and 1.5 TB of memory, with your Hydra home directory, `/data`, `/scratch` and `/store` mounted. It runs R 4.5.3 and RStudio Server 2024.12.0+467, the same for every user, and needs no tunnel and no `qrsh`.

Log in

1. From the SI network or the SI VPN, open <https://galaxy.si.edu/R4> in the browser:



From telework.si.edu, enter `https://galaxy.si.edu/R4` in the **Enter an internal resource** box on the telework home page and press Return:



2. Sign in with your Hydra username, in lower case, and your Hydra password:

Server 762: 192 CPU cores, 1.5TB Memory
RStudio version: 2024.12.0+467
R version: R-core-4.4.3_1.el8.x86_64

Hydra Userid:

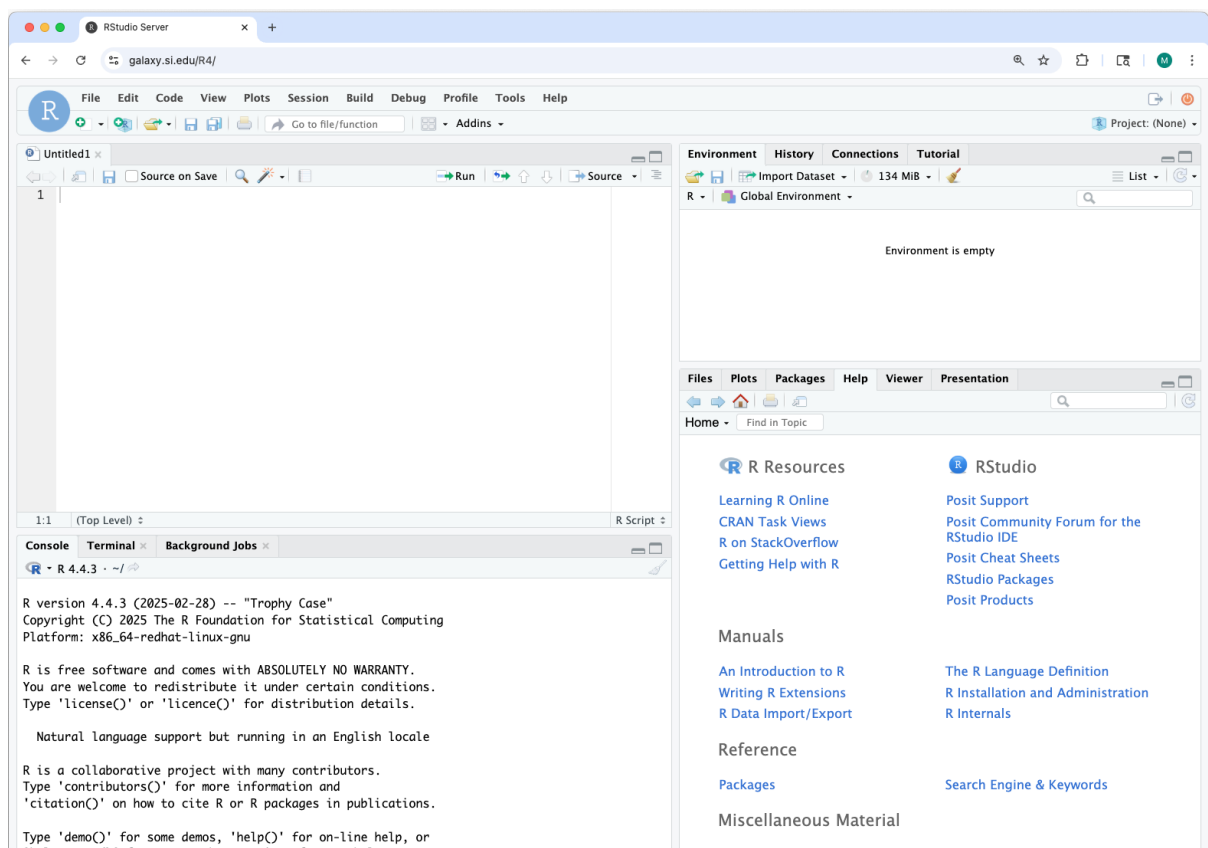
Password:

☒ Stay signed in when browser closes

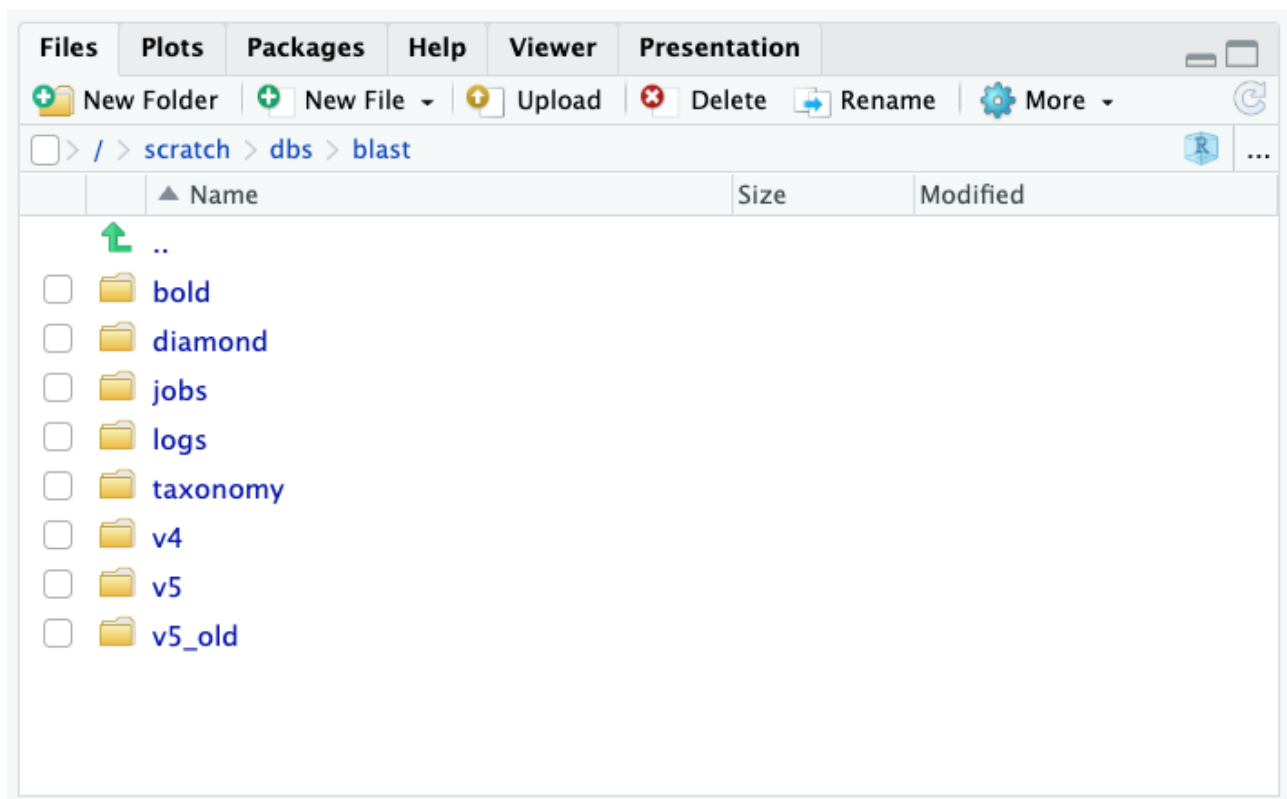
You will automatically be signed out after 60 minutes of inactivity.

Sign in

3. RStudio opens in the browser with the same layout as the desktop version:



The **Files** pane shows Hydra's partitions. Navigate to a directory under `/scratch` or `/data` to work there:

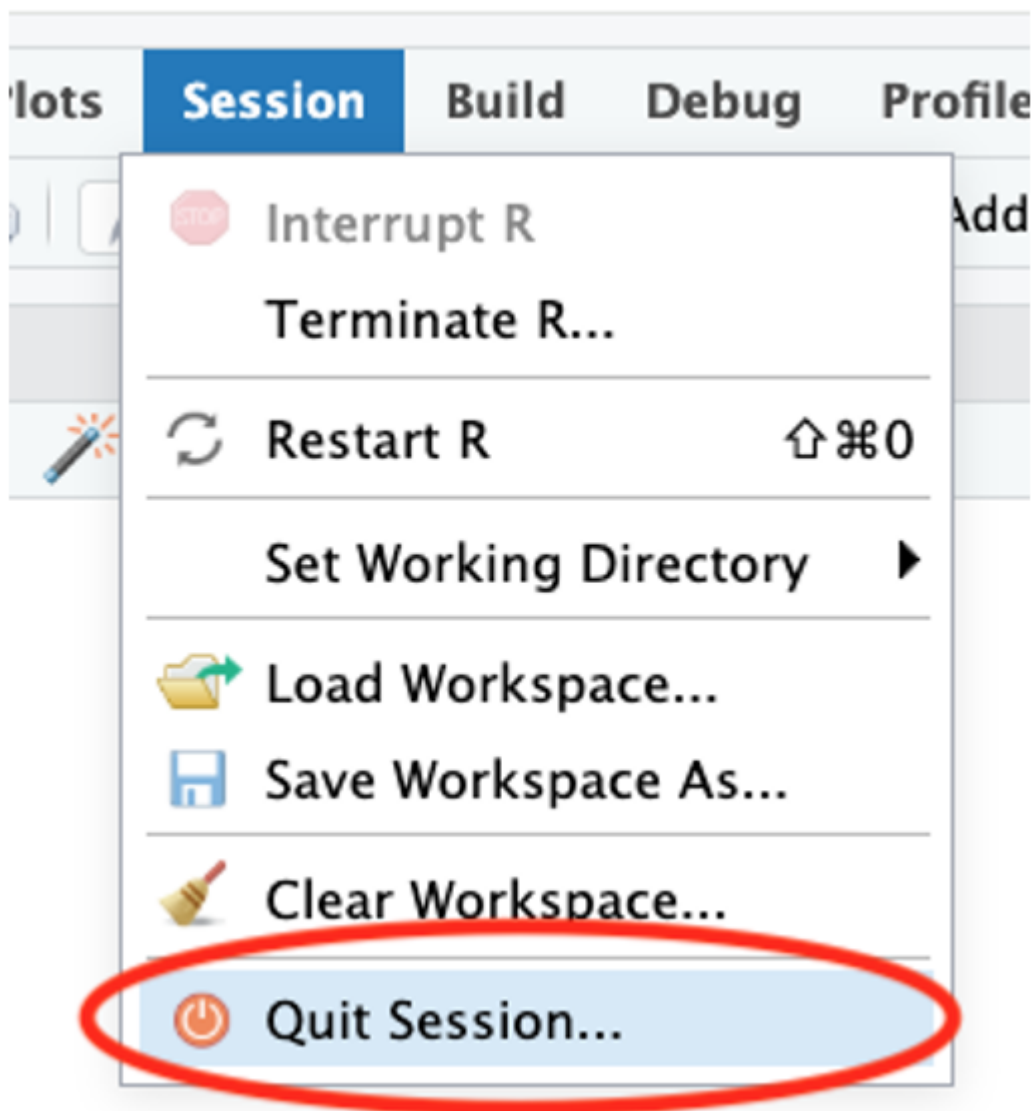


Sessions

The node is shared with every other user. One R session per user runs at a time. Logging in from a second browser takes over the existing session rather than starting another. A session keeps running when you close the browser or log off your computer, with its objects in memory and its computations under way. Logging in again reconnects to it. To run several analyses at once inside the one session, use the **Background Jobs** tab ([Posit's guide](#)).

Quit the session when you are done

An idle session holds memory that other users cannot get. **Session** and then **Quit Session...** ends it. Closing the browser does not.

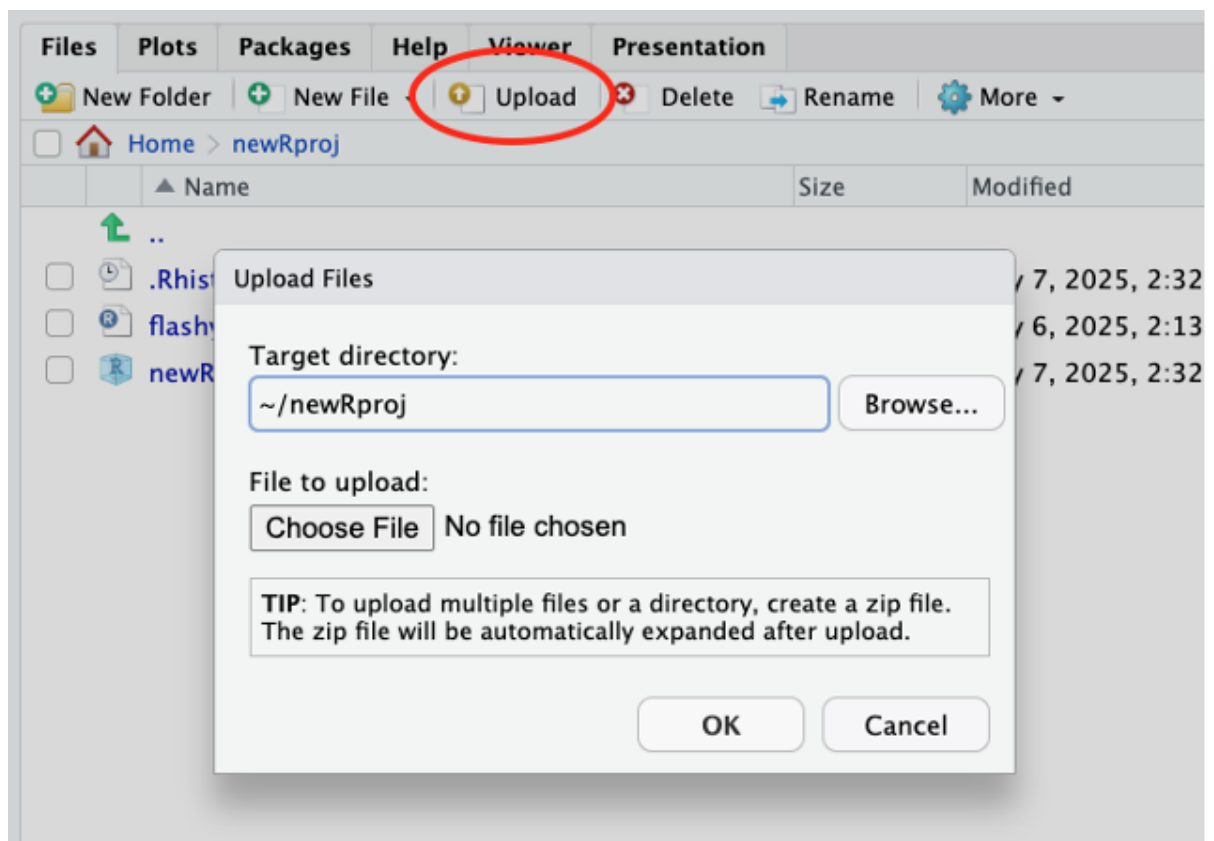


If the server refuses to log you back in after you used its log-off button, clear the browser's cookies for galaxy.si.edu.

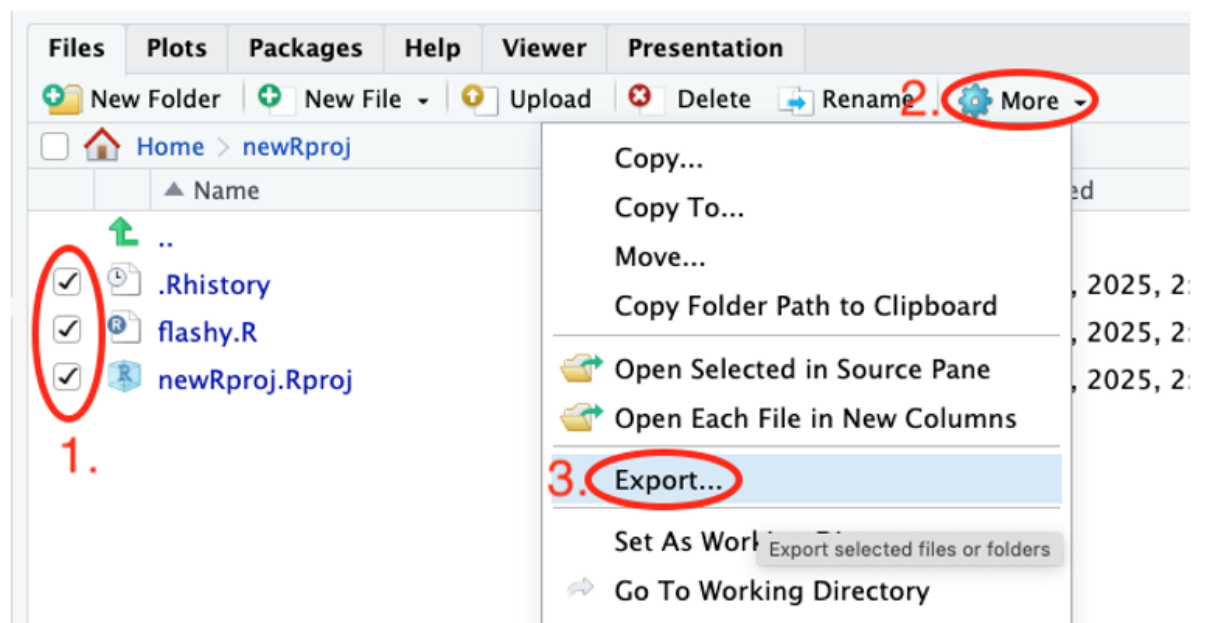
Move files in and out

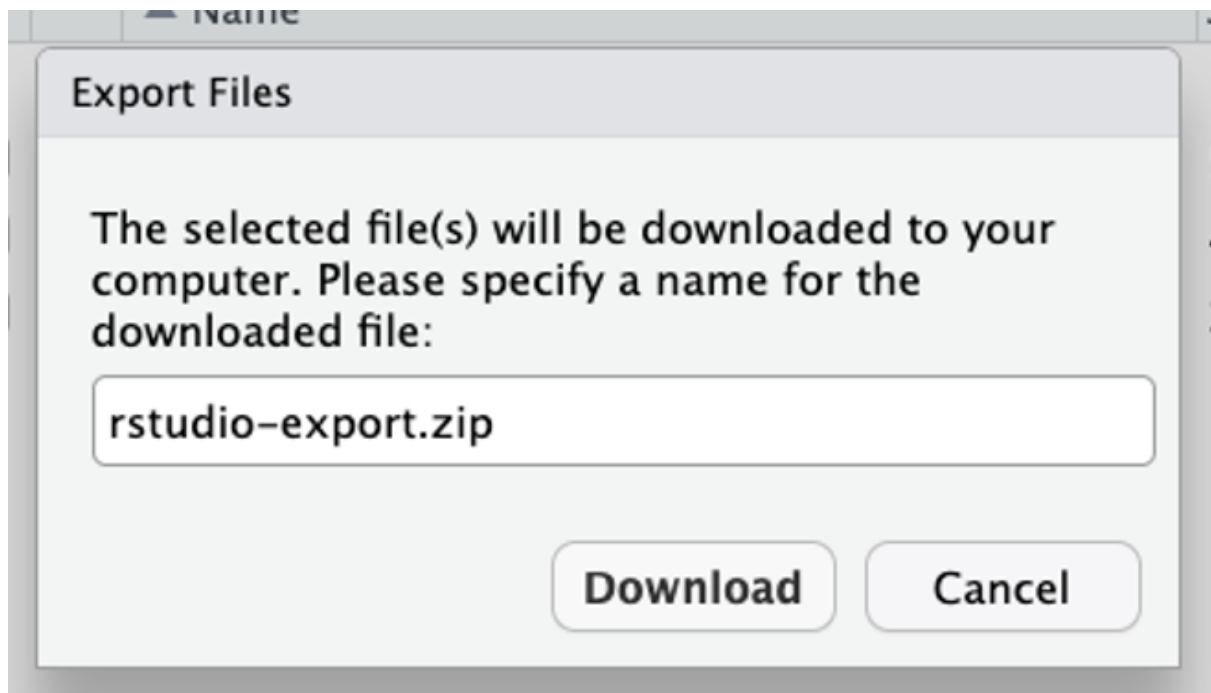
The **Files** pane transfers files. For anything large or many files, use [Data transfer](#) instead.

- Upload: **Upload** in the Files pane, one file at a time. To upload several, zip them on your computer. The server unzips the archive on arrival.



- Download: tick the files or folders, then **More** and **Export...**. Several files or a folder come down as one zip.





Install packages

`install.packages()` and `BiocManager::install()` work as on a desktop, into your personal library at `~/R/x86_64-redhat-linux-gnu-library/4.5`, which other users do not see. Every package compiles from source on Linux, so installs take minutes. `Ncpus` parallelises them:

```
install.packages("seqinr", Ncpus = 8)
BiocManager::install("phyloseq", Ncpus = 8)
```

Packages that need a newer GCC

Some packages (`ade4` and `seqinr` among them) need a newer compiler than the server's system GCC and fail with missing `GLIBCXX` symbols or OpenMP errors. The server cannot load compiler modules, so point R at one of Hydra's GCC installations with a `~/R/Makevars` file.

1. If `~/Renvirom` sets compiler paths, remove it. Mixed compiler settings are the usual cause of the errors:

```
$ rm ~/Renvirom
```

2. Create the Makevars file, in the RStudio **Terminal** (not the R console):

```
$ mkdir -p ~/.R
$ nano ~/.R/Makevars
```

`~/R/Makevars`

```
CC = /share/apps/tools/gcc/14.2.0/bin/gcc
CXX = /share/apps/tools/gcc/14.2.0/bin/g++
CXX11 = /share/apps/tools/gcc/14.2.0/bin/g++
CXX14 = /share/apps/tools/gcc/14.2.0/bin/g++
CXX17 = /share/apps/tools/gcc/14.2.0/bin/g++
FC = /share/apps/tools/gcc/14.2.0/bin/gfortran
F77 = /share/apps/tools/gcc/14.2.0/bin/gfortran

CFLAGS = -O2 -g -fopenmp -fpic
CXXFLAGS = -O2 -g -fopenmp -fpic
CXX11FLAGS = -O2 -g -fopenmp -fpic
CXX14FLAGS = -O2 -g -fopenmp -fpic
CXX17FLAGS = -O2 -g -fopenmp -fpic
FFLAGS = -O2 -g -fpic
FCFLAGS = -O2 -g -fpic
```

Every line names the same GCC. 14.2.0 can be any of 9.2.0, 9.3.0, 10.1.0 (OpenMP 4.5), 11.2.0, 12.2.0 (OpenMP 5.0), or 13.2.0, 14.2.0, 15.2.0 (OpenMP 5.1). The ade4 package needs 10.1.0 or newer.

3. **Session** and then **Restart R**. The compiler settings take effect in the new session.
4. Install as usual.

To go back to the system GCC, delete `~/R/Makevars` and restart R. A package that installs but fails to load points at a Makevars file in which the lines name different GCC versions.

Temporary files

R writes temporary files under `/tmp` on the node, which is small and shared. Put them under your own space by setting `TMPDIR` in `~/Renvirom`, or in `.Renvirom` in the project directory. A shell `export` has no effect, because the server's R session does not read your shell's environment:

`~/Renvirom`

```
TMPDIR=/scratch/genomics/USERNAME/tmp
```

Further reading

- [R](#) for the R modules, packages and how to run R in a batch job
- [RStudio on a compute node](#) for a different R or more than one session
- [Storage](#) for where to keep data and projects

Last updated 2026-10-03

RStudio on a compute node

RStudio can also run on a compute node of your choosing, with the CPUs and memory of an [interactive session](#), served to your browser through an ssh tunnel. Use it when the [RStudio server](#) does not fit, which means a different R, more than one session, or CPUs and memory scheduled to you alone.

Start RStudio in a session

1. On a login node, start an interactive session and load the module:

```
$ qssh -pe mthread 4
$ module load tools/R/RStudio/server
```

2. The first time only, create the server's configuration file:

```
$ init-rstudio-server
init-rstudio-server: db.conf file created:
-rw-rw-r--. 1 USERNAME USERNAME 433 Oct  3 16:24 /home/USERNAME/.local/share/
rstudio/rstudio-server/db.conf
```

Without it the server does not start, and says `db.conf not found`, you need to run `init-rstudio-server` first.

3. Start the server:

```
$ start-rstudio-server -port 8123
start-rstudio-server: starting RStudio server on host=compute-64-11 and
port=8123
you need to create a ssh tunnel on your local machine with
ssh -N -L 8123:compute-64-11:8123 USERNAME@hydra-login01.si.edu

Point your browser to http://localhost:8123 on your local machine.
Use Control+C in this window to kill the server when done.
```

`-port` chooses the port, and the default is 8787. `Address already in use` means another user has that port on the node. Choose another, between 1025 and 65535. `start-rstudio-server -help` lists the options.

4. On your own machine, open the tunnel the script printed, in a terminal you leave alone:

```
$ ssh -N -L 8123:compute-64-11:8123 USERNAME@hydra-login01.si.edu
```

5. Open `http://localhost:8123` in a browser and sign in with your Hydra username and password.

6. When done, sign out in the browser, `Ctrl-C` in the server's window, `Ctrl-C` in the tunnel's terminal, then `exit` the session. Do not leave a server running unattended.

The module loads R 4.4.1 (`tools/R/4.4.1`). `module load tools/R` gives 4.4.0. If a sign-out leaves you unable to sign back in, stop and restart the server. If that fails, clear the browser's cookies.

RStudio desktop over X11

`tools/R/RStudio/desktop` provides the windowed RStudio, drawn on your machine through X11. It needs an X server on your machine and an `ssh -Y` connection, is slow, and does not work with Xming on Windows (Cygwin/X and WSL work). Start it from a `qlogin` session with `rstudio --no-sandbox --disable-gpu`.

Last updated 2026-10-03

Storage

Hydra has four kinds of disk space. Keep scripts and configuration in `/home`, results in `/data`, the working files of jobs in `/scratch`, and project data waiting for analysis in `/store`. Each has its own quota and retention rules. See [Filesystems](#).

- To see how much space you are using, [check your usage and quotas](#).
- To get back a file you deleted or overwrote on `/home`, `/data` or `/store`, [recover it from a snapshot](#).
- To get back a file the scrubber removed from `/scratch`, [request a restore](#) within about ten days.
- For a job that reads and writes intensively, [use the node's local SSD](#).
- For data on `/store`, [copy it with the I/O queue](#) before a job uses it.

Files on `/scratch/public` older than 180 days are deleted every week

`/scratch` has no snapshots and no backup. Keep results on `/data` or off Hydra. We can restore a scrubbed file for about ten days after the [scrubber](#) removes it.

None of the storage systems on Hydra are meant for archival storage. [Backups](#) explains what is protected against what, so that you can decide where else to keep a copy. To move data on and off the cluster, see [Data transfer](#).

Last updated 2026-10-03

Filesystems

Hydra has public partitions and a number of project partitions, each with its own quotas and retention rules. We adjust sizes and quotas as needed. `disk-usage -d all+ -quotas` on a login node prints the current values ([Check your disk usage and quotas](#)).

Public partitions

Partition	System	Size	Space quota, soft / hard	File quota, soft / hard
/home	NetApp	22 TB	350 GB / 384 GB	9 M / 10 M
/data/public	NetApp	330 TB	4.3 TB / 4.5 TB	9.5 M / 10 M
/scratch/public	GPFS	800 TB	14 TB / 15 TB	37 M / 39 M

Partition	Snapshots	Scrubbed	Use for
/home	4 weeks	no	configuration files, scripts, job files, source code
/data/public	2 weeks	no	final results, small important files; space from deleted files is not freed until the snapshots age out
/scratch/public	none	files older than 180 days, weekly	input data and working files for jobs; the fastest partition, over InfiniBand

`/store/public` is being phased out

`/store/public` is no longer available. Use `/data/public`. A cloud-based cold-storage service is being evaluated as the replacement. Project `/store` partitions are not affected.

A soft limit can be exceeded for a grace period. At the hard limit writes fail. The file quota counts inodes, so many small files reach it before the space quota does. `/scratch/dbs` (10 TB) holds the shared bioinformatics databases. See [Local databases](#).

`/scratch/public` is divided by unit or discipline into `biology`, `genomics`, `humanities`, `nasm`, `odi` and `sao`. Your directory is under one of them, `/scratch/public/genomics/USERNAME` for example. The shorter form `/scratch/genomics` is a link to `/scratch/public/genomics`, and either works. On `/data`, `/data/genomics` is the same directory as `/data/public/genomics`.

Project partitions

Groups with their own funding have dedicated partitions under `/scratch` and `/store`, with quotas set per project. `quota+` shows the ones you have access to. Dedicated space is bought with project funds when the disk farm is expanded. Email SI-HPC@si.edu.

Storage systems

System	Partitions	Access	Properties
NetApp	<code>/home</code> , <code>/data</code>	every node	snapshots; disaster-recovery copy to AWS Glacier (see Backups)
GPFS over InfiniBand	<code>/scratch</code>	every node	fastest; no snapshots; scrubbed
NAS	<code>/store</code> (project partitions)	login, head, interactive and RStudio server nodes only	snapshots on most partitions, for 2 to 8 weeks; fault tolerant but not highly available (see Use /store and the I/O queue)
local SSD	<code>\$SSD_DIR</code>	the node the job runs on, while it runs	see Use a node's local SSD

Scrubbing

Files on `/scratch/public` older than 180 days are deleted every week

`/scratch` has no snapshots and no backup. Move anything you need to keep to `/data` or off Hydra before it is 180 days old.

A scrubber runs weekly and removes files on `/scratch/public` older than 180 days, and old empty directories. It holds the removed files in a staging area for about ten days before deleting them. During that time they still count against your quota, and we can restore them on request. See [Find scrubbed files and request a restore](#). A restored file's change time (`ctime`) is reset, so it is safe for another 180 days.

None of the storage systems on Hydra are meant for [archival storage](#). Delete what you no longer need. Compress or archive sets of small files. `tar -czf archive.tgz dir/` packs a directory into one file, and `tar -xf archive.tgz` unpacks it.

Last updated 2026-10-03

Check your disk usage and quotas

Every partition has a per-user quota on space and on the number of files ([Filesystems](#) lists them). `quota+` shows where you stand, `disk-usage` how full each partition is, and `du-report` which directories hold the most.

Check your quotas

1. Print your usage and quotas on every partition:

```
$ quota+
Disk quotas for user USERNAME (uid 12345):
Mounted on          Used    Quota   Limit   Grace   Files
Quota   Limit   Grace
-----
-----
/data/public          2.06T   4.28T   4.50T       0   5.47M
9.50M  10.00M       0
/home                71.14G  350.0G  384.0G       0  667.2k
9.00M  10.00M       0
/scratch/public       5.16T  14.00T  15.00T       0   3.35M
37.75M  39.85M       0
/store/PROJECT        1.00G   none    none
```

`Quota` is the soft limit and `Limit` the hard limit, for space and for the number of files. `quota+ +%` adds the percentage used. `quota+ -f /scratch/public` shows one partition. The Linux `quota` command works only on `/home` and `/data`.

2. Delete or archive files on any partition where `Used` is close to `Limit` or `Files` is close to its `Limit`.

Jobs fail when a partition reaches its hard limit

Every write to that partition fails, including the log file of a running job. Free space before submitting more jobs.

A [warning email](#) goes out when your usage passes 95% of a quota.

See how full a partition is

```
$ module load tools/local
$ disk-usage -d all+
Filesystem                                Size      Used      Avail Capacity  Mounted on
netapp-fas83:/vol_home                    22.36T    17.19T     5.18T    77%/12%    /home
netapp-fas83-n01:/vol_data_public         332.50T   176.06T   156.44T   53%/2%    /data/
public
gpfs02:public                            800.00T   511.44T   288.56T   64%/28%    /scratch/
public
...
```

`Capacity` is space used / files used. `disk-usage -d all+ -quotas` adds the default quotas of each partition. `df -h /scratch/public` gives the same for one partition without the tool. The [status page](#) plots the same figures over time under Disk Usage & Quota.

Find what takes the space

1. Load `tools/local+` and run `dus-report` on the directory. It runs `du`, which takes minutes on a large tree, and lists the biggest subdirectories:

```
$ module load tools/local+
$ dus-report /scratch/genomics/USERNAME
612.372 GB      /scratch/genomics/USERNAME
                capac.  14.000 TB (4% full), avail.  13.388 TB
447.026 GB  73.00 % /scratch/genomics/USERNAME/project-a
308.076 GB  50.31 % /scratch/genomics/USERNAME/project-a/v4
138.950 GB  22.69 % /scratch/genomics/USERNAME/project-a/vX
report in /tmp/dus.scratch.genomics.USERNAME.USERNAME
```

2. Rerun on the saved report to see smaller directories without running `du` again:

```
$ dus-report -n 999 -pc 1 /tmp/dus.scratch.genomics.USERNAME.USERNAME
```

`-pc 1` lists everything down to 1% of the total. `-n` caps the number of lines.

`du -sh dir/` gives the size of one directory. `quota+` is available in every session. `disk-usage` needs `module load tools/local` and `dus-report` needs `module load tools/local+`. `man quota+`, `disk-usage -help` and `dus-report -help` list their options.

Reduce your usage

1. Delete input and intermediate files from finished jobs.
2. Compress large files that are not being read, with `gzip FILE`.
3. Archive a directory of small files into one file, then delete the directory:

```
tar -czf project-a.tgz project-a/  
rm -rf project-a/
```

This frees inodes as well as space. `tar -xf project-a.tgz` unpacks it.

4. Move data that is finished with [off Hydra](#).

On `/data` and `/home`, space from deleted files is not freed until the snapshots holding them age out, after 2 and 4 weeks. See [Recover a file from a snapshot](#). `/scratch` frees it at once.

Further reading

- [Filesystems](#) for the quota of each partition and what it is for
- [Recover a file from a snapshot](#) for why deleted files on `/home` and `/data` do not free space at once
- [Data transfer](#) for moving results off the cluster

Last updated 2026-10-03

Recover a file from a snapshot

`/home`, `/data` and the project `/store` partitions keep read-only snapshots of their contents, for 4 weeks on `/home` and 2 weeks on `/data`. On `/store` the period depends on the partition. You copy a file you deleted or overwrote within that time back from the snapshot.

`/scratch` has no snapshots

A file you delete from `/scratch` cannot be recovered. We can restore only files the `scrubber` removed, and only for about ten days.

Recover a file on `/home` or `/data`

1. List the snapshots of the partition. They are in a hidden `.snapshot` directory at its top, named by frequency and time:

```
$ ls /data/genomics/.snapshot
daily.2026-10-02_0010    hourly.2026-10-03_1005  hourly.2026-10-03_1305  weekly.
2026-09-27_0015
daily.2026-10-03_0010    hourly.2026-10-03_1105  hourly.2026-10-03_1405
hourly.2026-10-03_0905  hourly.2026-10-03_1205  weekly.2026-09-20_0015
```

2. Change into the snapshot from before the loss, at the path the file had:

```
cd /data/genomics/.snapshot/daily.2026-10-02_0010/USERNAME/analysis/results
```

3. Copy the file back. `-p` keeps its dates. `-i` asks before overwriting an existing file:

```
cp -pi results.csv /data/genomics/USERNAME/analysis/results/results.csv
```

To keep the current version as well, copy the old one under another name:

```
cp -pi results.csv /data/genomics/USERNAME/analysis/results/results-old.csv
```

Files under `.snapshot` can be read and copied (with `cp`, `tar` or `rsync`) but not moved or deleted. `/home` works the same way, under `/home/.snapshot/`.

Recover a file on `/store`

`/store` snapshots are under `.zfs/snapshot` at the top of each `/store` partition, named `auto-YYYY-MM-DD_HH-MM`:

```
$ ls /store/PROJECT/.zfs/snapshot
auto-2026-09-20_03-00  ...  auto-2026-10-03_03-00
$ cp -pi /store/PROJECT/.zfs/snapshot/auto-2026-10-03_03-00/USERNAME/data.tar /store/
PROJECT/USERNAME/
```

How far back `/store` snapshots go depends on the partition

Some partitions keep 8 weeks of snapshots, some keep 2 or 4 weeks, and some list none. List the directory on your own partition to see what it has before you rely on it.

`/store` is mounted only on the login nodes, the head node, the interactive nodes and the RStudio server node.

A snapshot lives on the same storage system as the partition, so it does not protect against a failure of that system. [Backups](#) describes what does.

Further reading

- [Backups](#) for the disaster-recovery copy and what it is for
- [Filesystems](#) for the snapshot schedule of each partition

Last updated 2026-10-03

Find scrubbed files and request a restore

Every week the scrubber removes files older than 180 days, and old empty directories, from `/scratch/public`. It moves the files to a staging area and deletes them permanently about ten days later. The rules are under [Filesystems](#). You receive an email when any of your files are scrubbed. You can check beforehand what the scrubber will remove, find out afterwards what it removed, and ask for some of it back.

The scrubber tools are in the `tools/scrubber` module. `module help tools/scrubber` lists them. Each has a man page.

Check what will be scrubbed

1. Load the module and search a directory for files near the age limit:

```
$ module load tools/scrubber
$ find-scrub -in /scratch/genomics/USERNAME/project-a -age 173
```

`-in` defaults to the current directory. `-age` defaults to 173 days, a week before the limit.

2. Move anything you still need to `/data`, or off Hydra, before the next Sunday.

`find-scrub` reads every file in the tree, which loads the file server. Run it only on the directories you care about.

Find out what was scrubbed

1. Note the date in the email. The report is keyed by it as `YYMMDD`.
2. Print the report for your directory and that date:

```
$ module load tools/scrubber
$ show-scrubber-report /scratch/public/genomics/USERNAME 260920
```

3. List the scrubbed files, or the scrubbed empty directories:

```
$ list-scrubbed-files -long /scratch/public/genomics/USERNAME 260920
$ list-scrubbed-dirs -long /scratch/public/genomics/USERNAME 260920
```

`-long` adds age and size. `-all` adds the owner. `-n` prints only the count. A Perl regular expression as a last argument limits the list, for example `'^/scratch/public/genomics/USERNAME/project-a/.*.log$'` for the `.log` files under `project-a`.

Request a restore

We restore files only while they are in the staging area, and only for a list you have trimmed to what you need.

1. Write the list of scrubbed files under the directory you want back:

```
$ module load tools/scrubber
$ list-scrubbed-files /scratch/public/genomics/USERNAME 260920 /scratch/public/
genomics/USERNAME/project-a > restore.list
```

The path argument matches as a prefix, so `project-a` also matches `project-a2`.

2. Edit `restore.list` down to the files you need.
3. Verify the list:

```
$ verify-restore-list -d /scratch/public/genomics/USERNAME 260920 restore.list
```

Fix anything it reports and verify again. One list per scrubbing date and per partition.

4. Email the full path of the list file to SI-HPC@si.edu before 5 pm on the Friday after the scrubbing. Send the path. Do not attach the file.

Scrubbed files are permanently deleted about ten days after the scrubbing

After that no restore is possible. Files in the staging area still count against your quota until then.

A restored file gets a new change time (`ctime`), which is what the scrubber measures, so it is safe for another 180 days. `stat FILE` shows it.

Further reading

- [Filesystems](#) for the scrubbing rules
- [Backups](#) for what is and is not protected

Last updated 2026-10-03

Use a node's local SSD

Some compute nodes have a local solid-state disk that a job can use for its working files. It pays only for a job that reads and writes intensively. A job whose time goes to computation gains nothing and takes a limited resource. The SSD is visible only to the job, only while it runs, and only from the node it runs on, so the job copies its input in at the start and its results out at the end.

Everything on the SSD is deleted when the job is killed

The SSD is cleared when the job is killed, crashes or reaches a time limit, and the login nodes cannot see it. Copy results back to `/scratch` inside the job, and checkpoint long runs there.

Run a job on the SSD

1. Put the input the job needs, and only the I/O-intensive part, in one directory on `/scratch`, or in one compressed archive:

```
cd /scratch/genomics/USERNAME/project-a
tar -czf ../project-a.tgz .
```

Files the job only reads occasionally, and configuration files, can stay on `/scratch`.

2. Request the SSD space the job needs, as `-l ssd_res=SIZE`. The scheduler then places the job on a node with an SSD and reserves that much. The job cannot write more than it reserved.
3. In the job file, load `tools/ssd` and copy or unpack the input into `$SSD_DIR`, the directory the module points at:

```
module load tools/ssd
cd $SSD_DIR
tar -xf /scratch/genomics/USERNAME/project-a.tgz
```

4. Run the analysis with its input and output paths under `$SSD_DIR`. For a program that takes paths as options, use `$SSD_DIR` in the options. For one that reads them from a configuration file, write the file at run time from a template in which the path is a placeholder:

```
sed "s=XXXX=$SSD_DIR=" wow.gen > wow.conf
```

5. Copy the results back to `/scratch` and delete everything on the SSD:

```
cd $SSD_DIR
tar -czf /scratch/genomics/USERNAME/project-a-results.tgz output/ logs/
rm -rf *
```

The whole job file is:

wow.job

```
## -S /bin/sh
## -N wow -o wow.log -cwd -j y
## -l ssd_res=2560G
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
#
module load tools/ssd
module load special/wow
sed "s=XXXX=$SSD_DIR=" $HOME/wow/project-a.gen > $HOME/wow/project-a.conf
#
cd $SSD_DIR
tar -xf /scratch/genomics/USERNAME/project-a.tgz
mkdir output logs
wow --params=$HOME/wow/parameters.dat --config=$HOME/wow/project-a.conf -o $SSD_DIR/
output -l $SSD_DIR/logs
#
tar -czf /scratch/genomics/USERNAME/project-a-results.tgz output/ logs/
rm -rf *
#
echo = `date` job $JOB_NAME done
```

`~hpc/examples/ssd/test-ssd.job` is a runnable example. Writing results as one `.tgz` rather than moving files is faster when the data compresses, and takes less space on `/scratch`.

Save only new files

To copy back only what the analysis wrote, create a timestamp file before it starts and use `tar --newer` or `find -newer` afterwards:

```
date > $SSD_DIR/started.txt
wow ...
cd $SSD_DIR
tar --newer=$SSD_DIR/started.txt -czf /scratch/genomics/USERNAME/project-a-
results.tgz data/
rm -rf *
```

What happens to the SSD when the job ends

If the job ends normally with less than 50 GB left on the SSD, the node archives the leftover as a compressed tar file. With more, it deletes it. If the job is killed, the node deletes it. Two variables, passed with `-v`, change this:

Variable	Effect
SSD_SAVE_DIR=DIR	save the leftover into DIR, which must exist and be writable; - saves nothing
SSD_SAVE_MAX=SIZE	save only if the leftover is under SIZE (21.4M, 10G); 0 saves nothing

```
## -l ssd_res=10G -v SSD_SAVE_DIR=/scratch/genomics/USERNAME/save -v SSD_SAVE_MAX=10M
```

Keep `SSD_SAVE_MAX` under 80 GB. Saving more takes too long at job exit. Copy large results back in the job script instead.

Watch a job's SSD use

```
$ plot-qssduse JOBID          # PNG of the job's SSD use over time
$ plot-qssduse -x JOBID       # on screen, over X11
$ module load tools/local
$ plot-qssduse-summary        # SSD use across the cluster
```

`plot-qssduse-summary` needs `module load tools/local`. `JOBID.TASKID` selects one task of an array. `man plot-qssduse` has the options.

Further reading

- [Request a queue, memory and CPUs](#) for the other resources a job can request
- [Examples](#) for the tested SSD job under `~hpc/examples/ssd`

Last updated 2026-10-03

Use /store and the I/O queue

/store holds the project partitions on the NAS, a large and inexpensive near-line system. It holds data waiting to be processed or already processed, and it is not backed up. SI-HPC@si.edu sets up the groups that have a /store partition.

/store/public is being phased out

/store/public is no longer available. Use /data/public. A cloud-based cold-storage service is being evaluated as the replacement.

/store is mounted on the login nodes, the head node, the interactive nodes and the RStudio server node.

/store is not mounted on the compute nodes

A job that opens a path under /store fails. Copy the data to /scratch or /data first, on a login node or with an I/O job, and copy results back afterwards.

Copy data on a login node or in a qcrsh session

For a small copy, run `cp` or `rsync` on a login node, or under `qcrsh` for anything that takes more than a few minutes:

```
$ rsync -a /store/PROJECT/USERNAME/run-12/ /scratch/genomics/USERNAME/run-12/
```

The login-node limits apply (see [Warning emails](#)), and a `qcrsh` session ends after 48 hours.

Copy data as an I/O job

Large or repeated copies run as jobs in the I/O queue, `lTI0.sq`, which runs on the interactive nodes. An I/O job can run for 72 hours, and use 48 hours of CPU and 8 GB per slot. One user may run eight I/O jobs at once, with up to 8 slots in total.

1. Write a job file that requests the queue with `-q lTI0.sq -l ioq`:

getData.job

```
## -S /bin/sh
## -N getData -o getData.log -cwd -j y
## -q lTI0.sq -l ioq
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
rsync -a /store/PROJECT/USERNAME/run-12/ /scratch/genomics/USERNAME/run-12/
echo = `date` job $JOB_NAME done
```

2. Submit it:

```
$ qsub getData.job
Your job 15503728 ("getData") has been submitted
```

3. Chain the analysis and the copy back so they start when the previous step finishes:

```
$ qsub -hold_jid 15503728 analyze.job
Your job 15503729 ("analyze") has been submitted
$ qsub -hold_jid 15503729 saveNClean.job
Your job 15503730 ("saveNClean") has been submitted
$ qstat+ +a%
Total running (PEs/jobs) = 1/1, 2 queued (jobs) for user 'USERNAME'.
  jobID name                stat    age nPEs      cpu% queue
node taskID
  15503728 getData           r    00:01    1          lTIO.sq
64-14
  15503729 analyze          hqw    00:01    1
  15503730 saveNClean       hqw    00:01    1          lTIO.sq
```

`hqw` is a job waiting on a hold. `qchain getData.job analyze.job saveNClean.job` submits the three with the holds set. See [Run jobs in sequence](#).

`/store` snapshots and quotas are on the [Filesystems](#) page. `quota+` shows your `/store` usage, which the Linux `quota` command does not. Recovering a file from a `/store` snapshot is under [Recover a file from a snapshot](#).

Further reading

- [Filesystems](#) for the `/store` partitions, their quotas and snapshots
- [Queues](#) for the I/O queue's limits
- [Globus](#) for moving data between `/store` and other institutions

Last updated 2026-10-04

Backups

No partition on Hydra is backed up in a way you can restore from yourself. Two mechanisms exist, with different purposes. Snapshots are what you use to get back a recently deleted or overwritten file. The disaster-recovery copy of `/home` and `/data` exists to rebuild those partitions if the storage system fails.

Partition	Snapshots	Disaster-recovery copy	Scrubbed
<code>/home</code>	yes, 4 weeks	AWS Glacier	no
<code>/data</code>	yes, 2 weeks	AWS Glacier	no
<code>/scratch</code>	no	no	on <code>/scratch/public</code> , files older than 180 days
<code>/store</code> (project partitions)	on most partitions, 2 to 8 weeks	no	no
local SSD	no	no	deleted when the job ends

Snapshots

Use snapshots to recover a file you deleted or overwrote within the retention period.

[Recover a file from a snapshot](#) has the procedure for `/home`, `/data` and `/store`.

Disaster recovery

The Glacier copy of `/home` and `/data` is for rebuilding a partition after a storage failure. It is not for restoring individual files. A restore from Glacier takes staff time and is billed by Amazon, so a request needs a justification, and the requester may be asked to contribute to the cost. We keep the copies for up to a year.

Long-term storage

None of the storage systems on Hydra are meant for archival storage

Some projects keep data on Hydra for years, because of the volume they collect or because the data were here before other options existed, and their allocations reflect that. Identify a long-term destination off Hydra for every dataset that outlives its analysis, and move data there when the analysis ends. Moving data off the cluster is under [Data transfer](#).

Last updated 2026-10-03

Software

Hydra provides software through environment modules. You list what is installed with `module avail` and put one package on your path with `module load NAME/VERSION`. Most other packages you install yourself, into a conda environment in your own space.

- For a package that is installed, [find and load its module](#).
- For a Python package, or anything on conda-forge or Bioconda, [set up conda](#) and install it there.
- For software you built yourself, [write a module file](#) so that your jobs can load it.

We have a page each for [R](#), for [Julia](#), [MATLAB](#), [IDL](#) and [Java](#), for [GPUs](#) and for [containers](#).

Set the thread count from `$NSLOTS`

NumPy, R's linear-algebra libraries, IDL, Java and many other packages start one thread per CPU on the node unless told otherwise. A job that does this is oversubscribed and we can kill it. See [Warning emails](#). Each page below says how to set the count for that software.

When you need to look something up, the reference pages cover the [compilers](#), [libraries](#) and [MPI implementations](#) and the [installed modules](#). For a handful of packages we provide a [guide](#) with a working job file. If you need software that requires a license, a build against the cluster's MPI or CUDA libraries, or a shared database, email SI-HPC@si.edu.

Last updated 2026-10-03

Find and load software

A module file sets the environment (`PATH`, `MANPATH`, `LD_LIBRARY_PATH` and any variables the package needs) for one package and version, and works the same under `bash` and `csh`. You find a package, load it in a session or a job, and switch between versions with the `module` command. We maintain the modules for the packages the cluster provides. [Write a module file](#) covers your own.

Modules are grouped by prefix. `bio/` holds the bioinformatics packages (`bioinformatics/` is an alias), `tools/` general tools and languages, `gcc/`, `intel/` and `nvidia/` the compilers and their MPI builds, `idl/` and `matlab/` those runtimes, and `gis/` and `jupyter/` a few more. The [list of module files](#) is the complete inventory. The [installed modules](#) page lists the `bio/` and `tools/` prefixes.

Find a package

1. Search the module names:

```
$ module -t avail 2>&1 | grep -i samtools
bio/samtools/1.19.2(default)
```

`module -t avail` prints one module per line. `2>&1` is there because it writes to standard error. Without a pattern, `module avail` prints everything.

2. Read what the module provides and how to run it:

```
$ module whatis bio/samtools
$ module help bio/samtools
```

`module help` prints the executables the package installs and notes on running it on Hydra. `module show bio/samtools` prints what loading it changes in your environment.

Load a package

1. Load the module, with a version or without. Without a version you get the one marked (default):

```
$ module load bio/samtools
$ module load bio/samtools/1.19.2
```

2. Check what is loaded:

```
$ ml
Currently Loaded Modulefiles:
  1) uge/8.8.1   2) tools/local-user   3) bio/samtools/1.19.2
```

`uge/8.8.1` and `tools/local-user` are sticky. Every session loads them and they cannot be unloaded.

3. In a job file, put the same `module load` lines before the commands that use the package. Do not rely on modules loaded in your login shell, because the job does not inherit them (see [Do not use -v](#)).

`ml` is a shortcut. `ml` alone is `module list`, `ml bio/samtools` loads, and `ml -bio/samtools` unloads.

Switch or unload

Two versions of one package cannot be loaded at once. The module command reports a conflict. Switch versions instead of loading a second one, or unload first:

```
$ module switch gcc/12.2.0 gcc/13.2.0
$ module unload gcc/13.2.0
$ module load nvidia/24.3
```

`module purge` unloads everything except the sticky modules.

Change how module reports

Module	Effect
<code>module-nocolor</code>	no colors
<code>module-nowarn</code>	fewer warnings
<code>module-simple-format</code>	shorter <code>module list</code> output
<code>module-simple</code>	the three above
<code>module-verbose</code>	the same as <code>module -v</code>

Load or unload these like any other module. `tools/manpath` restores the default `man` page locations if loading a module has hidden them.

Perl, Python and CMake scripts can also call the module command. `man module` describes how. The Modules documentation at <https://modules.readthedocs.io/en/v5.3.1/> covers the version installed on Hydra.

Further reading

- [Installed modules](#) for the `bio/` and `tools/` packages and their versions
- [Write a module file](#) for software you built yourself
- [Job script reference](#) for why a job loads its own modules

Last updated 2026-10-03

Python and conda

Modules provide several Python installations, and conda installs anything else into your own space. What is specific to Hydra is which Python modules exist, how to keep NumPy and similar packages to the CPUs a job requested, and how to use conda in sessions and in jobs. How conda itself works is in the conda documentation at <https://docs.conda.io/>.

Use an installed Python

Module	Python	Note
<code>tools/python</code>	3.11.4, Anaconda	default; many packages preinstalled
<code>tools/python/3.13</code>	3.13.5, Anaconda	
<code>tools/python/3.14</code>	3.14.0	built from source, base packages only; run it as <code>python3</code>
<code>tools/python/3.9 , 3.10</code>	3.9.8, 3.10.0	built from source; run them as <code>python3</code>
<code>tools/python/3.7 , 3.8</code>	3.7.3, 3.8.8, Anaconda	
<code>tools/python/2.7</code>	2.7.16, Anaconda	
<code>intel/python/39-24.0</code>	3.9.18, Intel	with the Intel 2024.0 compilers

Run `module load tools/python` in a session or a job file. The three builds from source provide `python3` and no `python`. With one of them loaded, `python` is still the system's Python 3.6.8. `module -t avail 2>&1 | grep python` lists every version. The Anaconda builds include NumPy, SciPy, pandas, matplotlib and the rest of the Anaconda distribution. `pip list` after loading shows what is there. Packages you install with `pip install --user` go to `~/.local`, where only that Python version finds them.

Keep NumPy to the requested CPUs

NumPy, SciPy and other packages built on BLAS start one thread per CPU on the node unless told otherwise, which oversubscribes the node and gets the job killed (see [Warning emails](#)). Load one of two modules after the Python module:

Job	Load
serial, or MPI with one process per slot	<code>tools/python/use-single-thread</code>
<code>-pe mthread N</code>	<code>tools/python/use-multi-thread</code> , which sets the thread count to <code>\$NSLOTS</code>

numpy.job

```

## -S /bin/sh
## -N numpy -cwd -j y -o numpy.log
## -pe mthread 4
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
module load tools/python
module load tools/python/use-multi-thread
python crunch.py
echo = `date` job $JOB_NAME done

```

`module show tools/python/use-multi-thread` prints the variables it sets. The older names `tools/single-thread-numpy` and `tools/mthread-numpy` still work.

Use conda

There are two ways to get conda, and they do not mix. One is the preinstalled conda or mamba through a module. The other is a Miniconda you install yourself. Pick one. A `conda init` from one installation writes a block into `~/.bashrc` that breaks the other. If you switch, delete the block between `# >>> conda initialize >>>` and `# <<< conda initialize <<<`.

Run `conda install` under `qrsh`, not on a login node

Solving an environment uses a full CPU for minutes, which gets the process killed on a login node (see [Warning emails](#)). Start an [interactive session](#) first. `mamba` solves faster than `conda`.

Preinstalled conda or mamba

1. In a session, load the module and enable conda:

```

$ module load tools/conda
$ start-conda
(base) $

```

`tools/conda` loads Miniconda 23.1.0 by default. `tools/conda/25.9.1` is the newest, and `tools/conda/3.13` is the full Anaconda distribution. `tools/mamba` (default 25.9.1) is the same with `mamba` in place of `conda`. `module -t avail 2>&1 | grep -E 'tools/(conda|mamba)'` lists them.

2. Create an environment for each project or pipeline and install into it. Environments go under `~/.conda/envs`, which counts against your `/home` quota:

```

(base) $ conda create -n iqtrees-v1 -c bioconda -c conda-forge iqtrees=1
(base) $ source activate iqtrees-v1
(iqtrees-v1) $ iqtrees --version

```

Use `source activate` rather than `conda activate`, because it works in job files as well as in sessions.

3. In a job file, repeat the three lines:

iqtree.job

```
## -S /bin/sh
## -N iqtree -cwd -j y -o iqtree.log
## -pe mthread 4
## -l mres=8G,h_data=2G,h_vmem=2G
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
module load tools/conda
start-conda
source activate iqtree-v1
iqtree -s alignment.phy -nt $NSLOTS
echo = `date` job $JOB_NAME done
```

Adding `module load tools/conda` and `start-conda` to `~/.bashrc` enables conda in every session. The job file still needs its own lines.

Your own Miniconda

1. Install Miniconda under your home directory, following the installer's instructions at <https://docs.conda.io/en/latest/miniconda.html>. `/home` is not scrubbed, and a Miniconda with a few environments fits in its quota.
2. Write a [module file](#) that puts it on the path:

`~/modulefiles/miniconda`

```
##Module1.0
prepend-path PATH /home/USERNAME/miniconda3/bin
```

3. In a job file, load the module, then activate the environment:

```
module load ~/modulefiles/miniconda
source activate iqtree-v1
```

Packages without a conda recipe

Install into an environment with `pip` after activating it, so the package lands in the environment and not in `~/.local`:

```
(iqtree-v1) $ pip install PACKAGE
```

For software that needs a compiler, load the compiler module first (see [Compilers, libraries and MPI](#)). For software that needs a license, a build against the cluster's MPI or CUDA libraries, or a shared database, email SI-HPC@si.edu.

Jupyter

Jupyter runs on a compute node through an interactive session, with the notebook served to your browser through an ssh tunnel. See [Jupyter](#).

Further reading

- [Jupyter](#) for notebooks on a compute node
- [Submit a parallel job](#) for keeping NumPy and similar packages to the CPUs a job requested
- [conda documentation](#) from the conda project

Last updated 2026-10-04

R

R runs from the command line and in jobs through the `bio/R` module. Packages install into your own library, and R has to be kept to the CPUs a job requested. RStudio, on the RStudio server or on a compute node, is under [RStudio](#).

Load R

```
$ module load bio/R
$ R --version
```

`bio/R` loads R 4.4.0. `bio/R/4.4.1` selects another version, and `module -t avail 2>&1 | grep 'bio/R/'` lists them. `tools/R` and `bioinformatics/R` are aliases of `bio/R`. For a newer R than the modules provide, install it in a conda environment (see [Python and conda](#)) with `conda create -n r-env -c conda-forge r-base`.

Install packages

1. Start R on a login node or in an [interactive session](#) and install as usual:

```
install.packages("vegan")
```

R reports that the system library is not writable and offers to create a personal library under your home directory. Accept. Every package you install from then on goes there, and every job on every node sees it.

2. For a package that compiles C or Fortran code, install from a login node or an interactive session. Those nodes have the system development libraries such packages need and the compute nodes do not, so an install started inside a batch job fails at the compile step.

Use the requested CPUs

Give R functions that run in parallel the number of workers from `NSLOTS`, so the job uses the slots it requested however it was submitted:

```
numcores <- as.integer(Sys.getenv("NSLOTS"))
cl <- makeCluster(numcores, type = "FORK")
```

Start R workers with the FORK type

Use `type = "FORK"` on Hydra. Without it each worker is a separate R process with its own copy of the data, so the job uses far more resources than expected and may be killed.

R's linear-algebra libraries start one thread per CPU on the node unless told otherwise, which oversubscribes the node. Set `OMP_NUM_THREADS` in the job file before starting R:

Job	Set
serial	<code>export OMP_NUM_THREADS=1</code>
<code>-pe mthread N</code> , threads only	<code>export OMP_NUM_THREADS=\$NSLOTS</code>
<code>-pe mthread N</code> , all slots used by <code>makeCluster()</code>	<code>export OMP_NUM_THREADS=1</code>
<code>makeCluster()</code> with K workers plus threads	<code>export OMP_NUM_THREADS=\$((NSLOTS / K))</code>

model.job

```

#$ -S /bin/sh
#$ -N model -cwd -j y -o model.log
#$ -pe mthread 8
#$ -l mres=16G,h_data=2G,h_vmem=2G
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
module load bio/R
export OMP_NUM_THREADS=1
Rscript model.R
echo = `date` job $JOB_NAME done

```

R in a conda environment

Several jobs activating the same R conda environment at once can hang. The environment's `activate.d/activate-r-base.sh` runs `R CMD javareconf`, and concurrent runs overwrite each other's files. Either edit that script so the line reads

```
R CMD javareconf > /dev/null 2>&1 || true
```

or put the environment's `bin` directory on `PATH` in the job file instead of activating it, which works for most packages but skips the other variables activation sets.

For help, email SI-HPC@si.edu with the full error message and whether you were using the RStudio server, RStudio on a node, or R from the command line.

Further reading

- [RStudio server](#) for R in a browser on the dedicated node
- [Python and conda](#) for conda environments, which also work for R packages
- [Submit a parallel job](#) for jobs that use several CPUs

Last updated 2026-10-04

Julia, MATLAB, IDL and Java

Four languages need something Hydra-specific to run in a job, whether a module, a runtime license, or a start-up option that keeps the program to the memory and CPUs the job requested.

Julia

```
$ module load tools/julia
$ module load tools/julia/1.6.3
```

`module -t avail 2>&1 | grep julia` lists the versions, 1.2.0 to 1.12.1. The default is 1.10.2. `module help tools/julia` describes the build. In a job that requested `-pe mthread N`, start Julia with `julia --threads $NSLOTS`.

MATLAB

MATLAB itself is not installed, because there is no MATLAB license on Hydra. The MATLAB Runtime is, so a MATLAB program compiled elsewhere with the MATLAB Compiler runs here.

1. Compile the program with MATLAB Compiler on a machine that has it, for the runtime version you will load.
2. In the job file, load the matching runtime (`module load matlab/R2025b`) and start the compiled program through the launcher script the compiler produced. The module sets the runtime location the launcher needs. `~hpc/examples/matlab/README` shows a complete job.

`module -t avail 2>&1 | grep matlab` lists the runtimes, R2014a to R2025b. `matlab/rt` is the default.

IDL

Hydra has 5 interactive IDL licenses and 128 runtime licenses, for IDL 8.7 to 9.2. The interactive licenses are for preparing and checking code on a login node. Jobs use the runtime licenses, which the scheduler counts (see [Resource limits](#)).

1. Compile the procedure and everything it calls into a save file, on a login node:

```
$ module load idl/rt
$ idl
IDL> .run reduce
IDL> resolve_all
IDL> save, /routine, file='reduce.sav'
```

The save file is a snapshot, so recompile after every change to the code. Its name must match the top-level procedure.

2. Submit a job that requests a runtime license with `-l idlrt=1` and runs the save file with `-rt`:

reduce.job

```
#$ -S /bin/sh
#$ -N reduce -cwd -j y -o reduce.log
#$ -l idlrt=1
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
module load idl/rt
idl -rt=reduce.sav
echo = `date` job $JOB_NAME done
```

A procedure started with `-rt` takes no arguments. Pass parameters on standard input instead, and read them in the procedure when `n_params()` is zero:

```
idl -rt=make_my_model.sav << EOF
5943.
124.
sun
EOF
```

make_my_model.pro

```
pro make_my_model, temperature, density, name
  if n_params() eq 0 then begin
    temperature = 0.D0
    density = 0.D0
    name = ''
    read, temperature
    read, density
    read, name
  endif
  ; the computation goes here
end
```

Limit IDL to the CPUs the job requested

IDL starts as many threads as the node has CPUs. Put `CPU, TP00L_NTHREADS = 1` at the top of the procedure for a serial job, or set it to the slot count in a job that requested `-pe mthread N`.

`idl -rt=src/model` changes to `src` before running. Some IDL commands are refused in runtime mode. The IDL manual lists them. `qhost -F idlrt -h global` shows how many runtime licenses are free. `module -t avail 2>&1 | grep idl` lists the IDL versions.

FL, an open-source IDL-compatible interpreter with no license limits, is available as `tools/fl` and needs the same `CPU, TP00L_NTHREADS` setting.

Java

```
$ module load tools/java          # default
$ module load tools/java/21
```

Hydra has versions 8, 17, 18 and 21. Java reserves far more virtual memory than its heap. Under a virtual-memory limit (`h_vmem`) of 2 GB no installed version starts, with or without a heap option:

```
Error occurred during initialization of VM
Could not allocate compressed class space: 1073741824 bytes
```

Java 8 says `Could not allocate metaspace`. Under a limit of 4 GB every version starts. Give a Java job an `h_vmem` of 4 GB or more.

Java also sizes its heap for the whole node unless told otherwise. Start it with an explicit heap limit that fits inside the job's memory reservation:

```
java -server -XX:MaxHeapSize=1g -jar program.jar
```

Java uses more than the heap. Reserve memory for the job above `MaxHeapSize` (see [Reserve memory](#)). Oracle's documentation at <https://docs.oracle.com/en/java/javase/> lists the options.

Last updated 2026-10-04

GPUs

A job requests a GPU like any other resource, and the scheduler assigns it specific cards. A GPU helps only a program written for NVIDIA GPUs, through CUDA or a framework built on it. A program that does not use a GPU gains nothing from a GPU queue.

Hydra has 8 GPUs on three nodes:

Node	GPUs	GPU memory	CPU slots	Usable node memory
compute-50-01	4 × NVIDIA L40S	48 GB each	64	503 GB
compute-79-01, compute-79-02	2 × NVIDIA GV100 each	32 GB each	20	125 GB each

Request a GPU

1. Add the two required resources to the job. `gpu` admits the job to a GPU queue, and `ngpus=N` is the number of GPUs the job uses. The scheduler requires both. A job with `gpu` but no `ngpus` goes into `Eqw` and never runs.

```
#$ -q sTgpu.q
#$ -l gpu,ngpus=1
```

2. Add CPU slots on the same node if the program uses more than one CPU, with `-pe mthread Z`. Leave it off for a serial program. `ngpus` is per job, not per slot, so `-pe mthread 8 -l gpu,ngpus=2` is 8 CPUs and 2 GPUs.
3. Pick the queue by the time the job needs. `sTgpu.q` allows 7 hours of CPU time and 14 hours elapsed, `mTgpu.q` 6 and 12 days, and `lTgpu.q` 30 and 60 days, each with 64 GB resident and 256 GB virtual memory per slot. Memory limits multiply by `Z`, and the GV100 nodes have 125 GB usable in total, so a large per-slot request with several slots fits only on `compute-50-01`.
4. To require a specific card, add `gpuarch`:

```
#$ -l gpu,ngpus=1,gpuarch=L40S
```

Leave it off otherwise. The job then takes whichever GPU is free first.

A complete job file is:

`gpu.job`

```

## -S /bin/sh
## -N gpu -cwd -j y -o gpu.log
## -q sTgpu.q
## -pe mthread 4
## -l gpu,ngpus=1
## -l mres=32G,h_data=8G,h_vmem=8G
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
echo + GPUs: $SGE_HGR_GPUS, CUDA_VISIBLE_DEVICES=$CUDA_VISIBLE_DEVICES
module load nvidia
./julia-set -threads $NSLOTS
echo = `date` job $JOB_NAME done

```

For an interactive session with a GPU, in the `qgpu.iq` queue (12 h CPU, 24 h elapsed):

```
$ qcrsh -l gpu,ngpus=1
```

Delete `ngpu` without an `s` from old job files

Old job files may request `num_gpu` or its alias `ngpu`. That resource does nothing, and once we remove it the scheduler rejects a job that names it with `unknown resource`. Delete `num_gpu=...` or `ngpu=...` from the `-l` list and keep `ngpus=N`, with an `s`.

What the job gets

The scheduler assigns specific GPUs and sets two variables. `SGE_HGR_GPUS` holds the assigned devices as `gpu0 gpu1`, and `CUDA_VISIBLE_DEVICES` holds the same as `0,1`. CUDA programs and frameworks read `CUDA_VISIBLE_DEVICES` and see only those GPUs. The scheduler does not confine the job to them, so a program that addresses a GPU by absolute index instead of through `CUDA_VISIBLE_DEVICES` collides with another job's GPU.

The GPUs run in exclusive-process mode, in which each GPU serves one process at a time. A program that starts more processes than the job has GPUs fails with `all CUDA-capable devices are busy or unavailable`. Request CPU slots in proportion to the GPUs the job uses. A job that holds most of a node's CPUs with one GPU leaves the node's other GPUs unusable.

Limits

Queue	GPUs per user
<code>sTgpu.q</code>	4
<code>mTgpu.q</code>	3
<code>lTgpu.q</code>	2
<code>qgpu.iq</code>	1
all GPU queues together	4

`qquota -u $USER` shows your use against these. The full set is under [Resource limits](#).

Build GPU code

`module load nvidia` provides `nvcc` (CUDA C++) and the NVIDIA C, C++ and Fortran compilers, which also accept OpenACC directives and CUDA Fortran. `nvidia/YY/cuda` adds the full CUDA toolkit of release YY. `module -t avail 2>&1 | grep '^nvidia/'` lists them, 21.9 to 25.9.

The two GPU models take different drivers, which bounds the CUDA toolkit a program may be built with:

Node	Driver	CUDA toolkit up to
compute-50-01 (L40S)	595.71.05	13.2
compute-79-01, -02 (GV100)	550.54.15	12.4

CUDA 13 dropped the Volta architecture, so build code for the GV100 nodes with a CUDA 12.x toolkit. For one binary that runs on both cards, build with `-gencode arch=compute_70,code=sm_70 -gencode arch=compute_89,code=sm_89`. `~hpc/examples/gpu` holds CUDA, Python, IDL and MATLAB examples with their job files, and `~hpc/examples/gpu/cuda` the OpenACC and CUDA Fortran variants.

Watch a GPU

These commands run on a login node:

```
$ check-gpu-use           # GPUs in use and free on each GPU node
$ get-gpu-info 50-01 -d   # the GPUs on compute-50-01 and the processes using
                           them
$ qstat -s r -F gpuar,ngpus -q '*gpu*'   # card type and free GPUs per queue
instance
```

On the GPU node itself, inside a job or a `qcrsh` session, `nvidia-smi` reports the cards:

```
$ nvidia-smi
$ nvidia-smi dmon -d 30 -s pucm -o DT           # utilization every 30 s
$ nvidia-smi --query-gpu=name,index,memory.used,utilization.gpu --format=csv -l 15
```

`man nvidia-smi` is on the login nodes. `qacct+ -j JOBJID -show +gpus,gpu_usage` reports a finished job's GPU assignment and use (see [Monitoring tools](#)).

Further reading

- [Queues](#) for the GPU queues' limits
- [Cluster hardware](#) for the nodes and their GPUs
- [Examples](#) for the tested GPU jobs under `~hpc/examples/gpu`

Containers

The login, interactive and compute nodes have Singularity CE 4.4, with no module to load. You pull a container image on a login node and run it in a job.

1. Pull or copy the image on a login node, into a directory under `/scratch` or `/data`. Pulling downloads and converts the image, which takes minutes and up to several gigabytes:

```
$ cd /scratch/genomics/USERNAME/images
$ singularity pull docker://quay.io/biocontainers/samtools:1.19.2--h50ea8bc_0
```

2. Run it in a job. The container sees your home directory and the current directory. Bind other directories with `--bind`:

samtools.job

```
#$ -S /bin/sh
#$ -N samtools -cwd -j y -o samtools.log
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
singularity exec --bind /scratch /scratch/genomics/USERNAME/images/
samtools_1.19.2--h50ea8bc_0.sif samtools view -c reads.bam
echo = `date` job $JOB_NAME done
```

A container that uses a GPU needs `--nv` on the `singularity` command and a [GPU request](#) on the job. `~hpc/examples/containers` has working examples (`lolcow.job`, `test1.job`, `test2.job`) with their logs. The Singularity CE user guide at <https://docs.sylabs.io/guides/latest/user-guide/> covers the command itself.

Further reading

- [Examples](#) for the tested container job under `~hpc/examples/containers`
- [Singularity documentation](#) from the project

Last updated 2026-10-03

Write a module file

A **module file** is a short text file that sets the environment for one package. Write one for software you installed yourself, such as a package built under your home directory or a Miniconda, so that one `module load` line puts it on your path in a session or a job, in either shell.

Module files are written in Tcl, and most need three or four lines. The first line must be `##Module1.0`, and the rest set or extend environment variables. `man modulefile` lists every command.

Write the file

1. Create a directory for your module files:

```
mkdir -p ~/modulefiles
```

2. Write the file, named for the package. To put a Miniconda installation on the path:

~/modulefiles/miniconda

```
##Module1.0
prepend-path PATH /home/USERNAME/miniconda3/bin
```

To add a library directory:

~/modulefiles/geos

```
##Module1.0
prepend-path LD_LIBRARY_PATH /home/USERNAME/geos/lib
```

To set several variables from one base directory:

~/modulefiles/crunch

```
##Module1.0
set base /home/USERNAME/crunch
prepend-path PATH $base/bin
prepend-path LD_LIBRARY_PATH $base/lib
setenv CRUNCH $base
```

`set` makes a Tcl variable, used only inside the file. `setenv` exports a variable to the shell. `prepend-path` puts the directory first, `append-path` last. `remove-path`, `unsetenv`, `conflict` and `module load` are the other commands used in practice.

3. Load it by path and check the result:

```
$ module load ~/modulefiles/crunch
$ echo $CRUNCH
/home/USERNAME/crunch
```

Load it by name

1. Tell the module command where your files are, once, with a `~/.modulerc` file:

`~/.modulerc`

```
#%Module1.0
module use --append /home/USERNAME/modulefiles
```

2. Load by name from then on, in sessions and in job files:

```
module load crunch
```

The system module files are under `/share/apps/modulefiles` and are the best examples of what a module file can do. `module show NAME` prints what any of them does.

Last updated 2026-10-03

Compilers, libraries and MPI

Hydra has three compiler families, each with its own numerical libraries and MPI builds. When you need the module name for a compiler or an MPI, or the flag that turns on OpenMP, look it up here. Once the program is built, [Submit a parallel job](#) shows you how to run it.

Compilers

Family	Module	Compilers	Versions	Default
GNU	<code>gcc</code>	<code>gcc</code> , <code>g++</code> , <code>gfortran</code>	4.9.1 to 15.2.0	8.5.0
Intel oneAPI	<code>intel</code>	<code>icx</code> , <code>icpx</code> , <code>ifx</code> ; classic <code>icc</code> , <code>icpc</code> , <code>ifort</code> in older versions	2021.3 to 2025.3	2024.0
NVIDIA HPC SDK	<code>nvidia</code>	<code>nvc</code> , <code>nvc++</code> , <code>nvfortran</code> , <code>nvcc</code>	21.9 to 25.9	23.9

`module load gcc` loads the default. `module load gcc/13.2.0` loads a specific version. `module -t avail 2>&1 | grep '^gcc/'` lists every version of a family (`intel/`, `nvidia/` likewise). Compile, link and run a program with the same family and version, because libraries and runtimes from different compilers do not mix.

Intel renamed its compilers with oneAPI in 2021 (`icx`, `icpx`, `ifx`). The classic names remain in the versions that ship them. The NVIDIA compilers are the former PGI compilers. There is no separate PGI module. `nvcc`, the CUDA compiler, is part of the NVIDIA module. See [GPUs](#).

Libraries

Library	Provides	Examples
BLAS and LAPACK	linear algebra, one build per compiler family	<code>~hpc/examples/lapack</code>
Intel MKL	BLAS, LAPACK, FFT and more, with the Intel compilers	<code>~hpc/examples/lapack/intel</code>
GSL	GNU Scientific Library	<code>~hpc/examples/gsl</code>

The NVIDIA LAPACK build hangs or crashes in some cases. `~hpc/examples/lapack/nvidia/README` describes them.

MPI

Every MPI implementation has a build for each compiler family. Load the module that matches the compiler the program was built with and the implementation it was linked against. The module also defines `mpirun` for that build.

Module	Implementation	Parallel environment
gcc/V.R/openmpi, intel/YY/openmpi, nvidia/YY/openmpi	OpenMPI, default version; openmpi4 and openmpi5 select a major version, openmpi4.1.6-13.2.0 an exact build	-pe orte N
gcc/V.R/mvapich, intel/YY/mvapich, nvidia/YY/mvapich	MVAPICH 2, over InfiniBand	-pe mpich N
intel/YY/mpi	Intel MPI	-pe orte N
nvidia/YY/mpi	NVIDIA's bundled OpenMPI	-pe omp N

V.R is the GCC major and minor version (gcc/13.2/openmpi), and YY is the Intel or NVIDIA release year (intel/24/openmpi, nvidia/24/mvapich). The current module names are on the [Job script reference](#).

Build with the module loaded:

```
$ module load gcc/13.2/openmpi
$ mpicc -O2 -o hello hello.c          # mpif90 for Fortran, mpicxx for C++
```

In a job, the slot count comes from \$NSLOTS and the node list from \$PE_HOSTFILE (OpenMPI) or \$TMPDIR/machines (MVAPICH). See [Submit an MPI job](#). ~hpc/examples/mpi has a hello-world build for every compiler and implementation, described in its README.

OpenMP

Compiler	Flag
GNU	-fopenmp
Intel	-qopenmp
NVIDIA	-mp

An OpenMP program reads its thread count from OMP_NUM_THREADS. In a job, set it from the slots requested with -pe mthread N:

```
export OMP_NUM_THREADS=$NSLOTS
```

A multi-threaded program runs on one node, so its threads and memory are bounded by the largest node. An MPI program spans nodes. ~hpc/examples/openmp has an OpenMP build for each compiler. NVIDIA's compilers also accept OpenACC directives and CUDA Fortran for GPU code. See [GPUs](#).

Further reading

- [Submit a parallel job](#) for running the result under the scheduler
- [Examples](#) for a tested build and job file per compiler and MPI
- [Write a module file](#) for making your build loadable in jobs

Last updated 2026-10-03

Installed modules

This list is generated from `module -t avail`. Load a package with `module load PREFIX/NAME/VERSION`; `(default)` marks the version loaded when no version is given. The prefix `bioinformatics/` is an alias of `bio/`. The compilers and their MPI builds (`gcc/`, `intel/`, `nvidia/`) are under [Compilers, libraries and MPI](#); the [list of module files](#) (SI network or VPN) is the complete inventory, including versions this list omits.

bio/

Module	Versions
abyss	2.3.7(default)
admixtools	7.0.2(default)
admixture	1.3.0(default)
angsd	0.941(default)
assembly_stats	0.1.4(default)
aster	1.16, 1.19(default)
astral	5.7.8(default), 5.15.5-MP
augustus	3.5.0(default)
basespace	1.5.4, 1.6.2, 1.7.0(default)
bayescan	2.1(default)
bbmap	39.06(default)
bcftools	1.19(default)
beagle	4.0.1, 4.0.1-j8, 4.0.1-j8-cuda, 4.0.1-j17, 4.0.1-j17-cuda(default)
beast	2.7.5, 2.7.6(default), 2.7.7, 2.7.8
bedtools	2.31.1(default)
bioawk	1.0(default)
bioperl	1.7.8(default)
biopython	1.83(default)
birdnet	2.4.0(default), 2.4.0-gpu
blast	2.15.0(default), 2.17.0
blast2go	1.5.1(default)
blat	36(default)
blobtoolkit	3.3.10(default)
blobtools	1.1.1(default)
bowtie2	2.5.3(default)
bpp	4.7.0(default)
busco	5.7.0(default), 6.0.0
bwa	0.7.17(default)
cactus	2.8.0(default)
canu	2.2(default)
cd-hit	4.8.1(default)
cutadapt	4.7(default), 5.0
dates	753(default)
diamond	2.1.9(default)

Module	Versions
dorado	1.0.2(default)
eggno-mapper	2.1.13(default)
ensembl-vep	112.0(default)
exabayes	1.5.1(default)
exonerate	2.4.0(default)
fastp	0.23.4(default), 1.1.0
fastqc	0.12.1(default)
faststructure	1.0(default)
fasttree	2.1.11(default)
fastx_toolkit	0.0.14(default)
feems	1.0.0(default)
flye	2.9.5(default)
gatk	3.8.1.0, 4.5.0.0(default)
gemoma	1.9(default)
getorganelle	1.7.7.0, 1.7.7.1(default)
gone2	1.0.4(default)
gphocs	b807eb3(default)
graftm	0.15.1(default)
guppy	6.5.7(default)
hifiasm	0.19.8(default), 0.25.0
hisat2	2.2.1(default)
hmmer	3.4(default)
htslib	1.19.1(default)
hybpiper	1.3.1_final, 2.1.6(default), 2.3.3
illumiprocessor	2.10(default)
ipyrad	0.9.94(default)
iqtree	1.6.12, 2.3.1(default), 2.4.0, 3.0.1
jellyfish	2.3.1(default)
kentutils	v464
kraken	2.1.3(default)
krakenuniq	1.0.4(default)
mafft	7.525(default)
maker	3.01.03(default)
mapdamage2	2.2.2(default)

Module	Versions
masurca	4.1.1(default), 4.1.4
megahit	1.2.9(default)
mercury	1.3(default)
migrate	3.7.2, 5.0.6(default)
minimap2	2.28(default)
mitofinder	1.4.1(default)
mitos	2.1.8(default)
mitoz	3.6(default)
mrbytes	3.2.7a(default)
multiqc	1.23(default)
nextflow-qsub	2026-06-25(default)
ngsadmix	32(default)
ngstools	d7a78e4(default)
nucmer	3.2.3, 4.0.0rc1(default)
oma	2.6.0(default)
orthofinder	2.5.5(default)
paml	4.10.0, 4.10.7(default), 4.10.10
partitionfinder	2.1.1, UCE_SWSC(default)
patchwork	0.5.5
paup	4a168(default)
pcangsd	1.21(default)
phyluce	1.6.8, 1.7.3(default)
phynest	0.1.12(default)
picard-tools	2.20.6, 3.1.1(default)
plink	1.90b7.2(default)
psmc	0.6.5(default)
purge_dups	1.2.6(default)
qiime2	2024.2-amplicon(default), 2024.2-shotgun
quast	5.2.0(default)
R	4.4.0(default), 4.4.1, RStudio/desktop/2024.04.2-764, RStudio/server/2024.04.2-764
raxml	8.2.13(default)
raxml-ng	1.2.1(default)
ray	2.3.2-devel(default)

Module	Versions
relernn	1.0.0(default)
repeatmasker	4.1.5, 4.1.6(default), 4.1.7p1, 4.2.3
repeatmodeler	2.0.5(default)
revbayes	1.2.2(default), 1.3.0, 1.3.2, 1.4.1
rohan	1.0.1(default)
rsem	1.3.3(default)
samtools	1.19.2(default)
seqkit	2.8.1(default)
seqtk	1.4(default)
sharkmer	6f393f4(default), 97f19e2
smc++	1.15.2(default)
snippy	4.6.0(default)
spades	3.14.0, 3.15.5(default)
spruceup	v2022.2.4(default), v2024.8.20.post1
sratoolkit	3.1.0(default)
stacks	2.66(default)
stampy	1.0.31(default)
strauto	1.0(default)
structure	2.3.4(default)
tesseract	5.5.0(default)
transabyss	2.0.1(default)
transdecoder	5.7.1(default)
transrate	1.0.3(default)
treemix	1.13(default)
treepl	aaa8a0e(default)
treeshrink	1.3.9(default)
trimmomatic	0.39(default)
trim_galore	0.6.10(default)
trinity	2.15.1(default)
umi_tools	1.1.6(default)
vcftools	0.1.16(default)
whisper	20240930(default)
wtdbg2	2.5(default)

tools/

Module	Versions
awscli	2.15.27, 2.17.43(default), 2.31.36
bigtmp	
cmake	3.3.1, 3.24.2, 4.2.0
conda	3.8, 3.11, 3.13, 23.1.0(default), 23.11.0, 25.9.1
cpu_arch	
ffsend	0.2.74, 0.2.76(default), 0.2.77
fl	0.79.47, 0.79.50, 0.79.54
gnuparallel	20160422
gv	
java	1.8.0_45, 1.8.0_141, 17, 17.0.1, 18, 21, 21.0.2
julia	1.2.0, 1.5.3, 1.6.3, 1.10.0, 1.10.2(default), 1.12.1
jupyter	
local	
local+	
local-admin	
local-bc	
local-user <L:S>	
mamba	24.1.2, 25.9.1(default)
manpath	
misc	
mthread-numpy	
pigz	2.3.3(default)
python	2.7, 3.7, 3.8, 3.9, 3.9a, 3.10, 3.11(default), 3.11b, 3.13, 3.14, use-multi-thread, use-single-thread
python-mthread-numpy	
python-single-thread-numpy	
R	4.4.0(default), 4.4.1, RStudio/desktop/2024.04.2-764, RStudio/server/2024.04.2-764
rclone	1.62.2, 1.66.0(default), 1.71.2
ruby	2.2.2, 2.6.3
scrubber	
single-thread-numpy	
ssd	
svn	1.10.2

Module	Versions
vscode	1.91.0(default), 1.106.0
wget	1.21.4(default), 2.0.1, 2.2.0
xv	

Last updated 2026-09-24

Software guides

We provide a guide for each of the packages below. Each one tells you which module to load, which queue and how much memory to request, and gives you a job file to start from.

- [BEAST](#)
- [BirdNET](#)
- [BLAST](#)
- [Blast2GO](#)
- [BUSCO](#)
- [MaSuRCA](#)
- [SMC++](#)
- [SPAdes](#)
- [Local databases](#): BLAST, Dfam and Kraken databases kept on Hydra

If your package is not here, it may still be installed. [Installed modules](#) lists every `bio/` and `tools/` package, and `module help bio/NAME` on a login node tells you what a module provides and how to run it. If it is on Bioconda or conda-forge, you can install it yourself. See [Python and conda](#). If it needs a license, a build against the cluster's MPI or CUDA libraries, or a shared database, email SI-HPC@si.edu.

Last updated 2026-10-03

BEAST

BEAST 2 runs Bayesian phylogenetic analyses of molecular sequences. The module is `bio/beast`. `module -t avail 2>&1 | grep beast` lists the versions. BEAST's own guidance on performance is at <http://www.beast2.org/performance-suggestions/index.html>.

Version and packages

The first time BEAST runs it copies `beast.jar` and `beast.src.jar` into `~/.beast/2.x/BEAST/lib`, one subdirectory per version, and later runs load from there. Check the log file of every run for the version it reports, because a version left over in `~/.beast` can differ from the module you loaded.

BEAST packages (add-ons) install into `~/.beast` with the package manager, after loading the module:

```
$ module load bio/beast
$ packagemanager -list
$ packagemanager -add SNAPP
```

BEAGLE

The **BEAGLE** library is the module `bio/beagle`. `bio/beast` does not load it, so add `module load bio/beagle` after `module load bio/beast` in the job file. The default build, `4.0.1-j17-cuda`, is for Java 17 and BEAST 2.7.x, and is compiled with GPU and CPU support.

SNAPP path sampling as separate jobs

BEAST's [cluster instructions](#) for SNAPP path sampling produce one shell script per step, and each step runs as its own job on Hydra.

1. Add `doNotRun='true'` to the `run` element of the XML file.
2. In an interactive session with enough memory (`qcrsh -pe mthread 8`), load the module and start the XML with `-threads` set to the number of steps in the file, 48 in this example:

```
$ module load bio/beast
$ beast -threads 48 analysis.xml
```

This writes 48 step directories, each with a `runN.sh`.

3. Submit one job per step with this job file and loop:

```
beast_step_run.job
```

```

## -S /bin/sh
## -N beast_step_run -cwd -j y
## -q lThM.q
## -l mres=16G,h_data=16G,h_vmem=16G,himem
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
module load bio/beast
if [ -z "$1" ]; then
    echo "usage: qsub beast_step_run.job STEP"
    exit 1
fi
. ./run$1.sh
echo = `date` job $JOB_NAME done

```

submit.sh

```

#!/bin/sh
for x in $(seq 0 47); do
    qsub -o run$x.log beast_step_run.job $x
    sleep 0.1
done

```

Set the `seq` range to the number of steps (`run0.sh` to `run47.sh` here) and run `sh submit.sh` on a login node.

4. When every step has finished, run the analyser in an interactive session, adjusting `-alpha` and `-burnInPercentage` to the analysis:

```

$ qcrsh -pe mthread 8
$ module load bio/beast
$ applauncher PathSampleAnalyser -nrOfSteps 48 -alpha 0.3 -rootdir /scratch/
genomics/USERNAME/runs -burnInPercentage 25 > pathsampler.out 2>&1

```

`applauncher PathSampleAnalyser -help` lists the arguments.

Last updated 2026-10-03

BirdNET

BirdNET-Analyzer identifies bird species in audio recordings with a deep-learning model covering more than 6,000 species. Two modules provide version 2.4.0. `bio/birdnet` (the default) is for analysis, on CPUs, and `bio/birdnet/2.4.0-gpu` is for training custom classifiers on a GPU. Both provide `birdnet-analyze`, `birdnet-segments`, `birdnet-species`, `birdnet-train`, `birdnet-evaluate`, `birdnet-embeddings`, `birdnet-search` and `ffprobe`. The model comes with them, so jobs need no network access.

The models are licensed CC BY-NC-SA 4.0, for non-commercial use. Cite Kahl, Wood, Eibl and Klinck (2021), *Ecological Informatics* 61:101236.

Size the dataset

Analysis processes 30 to 60 hours of audio per CPU-hour, so the hours of audio divided by the slots requested gives the expected run time. The number of files matters only as the upper bound on useful slots. Count the hours with the module's `ffprobe`:

```
$ module load bio/birdnet
$ for f in /scratch/genomics/USERNAME/audio/*; do ffprobe -v error -show_entries form
at=duration -of csv=p=0 "$f"; done \
    | awk '{s+=$1} END {printf "%.1f hours in %d files\n", s/3600, NR}'
```

Analyse recordings

Analysis runs on CPUs. Do not request a GPU. `--threads` parallelises across files, one file per thread, so request no more slots than the directory has files. TensorFlow needs about 6 GB of virtual memory even for one file, so keep slots × `h_data` at 8 GB or more (for one slot, `-l mres=8G,h_data=8G,h_vmem=8G`).

birdnet.job

```
## -S /bin/sh
## -N birdnet -cwd -j y -o birdnet.log
## -q sThC.q
## -pe mthread 8
## -l mres=16G,h_data=2G,h_vmem=2G
#
set -e
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
echo + NSLOTS = $NSLOTS
module load bio/birdnet
birdnet-analyze /scratch/genomics/USERNAME/audio \
    -o /scratch/genomics/USERNAME/birdnet_out \
    --threads $NSLOTS \
    --min_conf 0.25
echo = `date` job $JOB_NAME done
```

`set -e` stops the job at the first failed command, so the final `done` line prints only on success. The example is at `/share/apps/bioinformatics/birdnet/2.4.0/examples/birdnet.job`.

For more than about 200 hours of audio, split the recordings into directories and run one [array task](#) per directory rather than one large job. Each recording is analysed independently, so the split costs nothing.

Train a custom classifier

Training runs on a GPU with the `2.4.0-gpu` module, as a job or under `qcrsh -l gpu,ngpus=1`, never on a login node. Training data is one subdirectory per class, each holding 3-second clips.

`birdnet-train.job`

```
## -S /bin/sh
## -N birdnet-train -cwd -j y -o birdnet-train.log
## -q sTgpu.q
## -l gpu,ngpus=1,gpuarch=L40S
## -l mres=16G,h_data=16G,h_vmem=64G
#
set -e
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
echo + CUDA_VISIBLE_DEVICES = $CUDA_VISIBLE_DEVICES
module load bio/birdnet/2.4.0-gpu
export OMP_NUM_THREADS=1
export TF_NUM_INTEROP_THREADS=1
export TF_NUM_INTRAOP_THREADS=1
birdnet-train /scratch/genomics/USERNAME/train_data \
    -o /scratch/genomics/USERNAME/custom_classifier
echo = `date` job $JOB_NAME done
```

`h_vmem` below 64 GB crashes training with a misleading message

TensorFlow and CUDA map about 40 GB of virtual memory for the smallest training run. With a smaller `h_vmem` the job fails at start with `cudaSetDevice ... out of memory`, which refers to the job's virtual-memory limit, not to GPU memory, and exits with status 0. `set -e` is what makes the job report the failure.

Give `-o` an absolute path. A relative path crashes when the classifier is saved. The three `export` lines keep TensorFlow's CPU thread pools inside the one slot the job requested. Training has run on the L40S node so far, which the `gpuarch=L40S` request selects. CPU slots can be added with `-pe mthread Z` (see [GPUs](#)), but training is GPU-bound and gains little from them. The example is at `/share/apps/bioinformatics/birdnet/2.4.0-gpu/examples/birdnet-train.job`.

To analyse recordings with the trained classifier, run `birdnet-analyze` from the CPU module, as above, with `--classifier` pointing at the saved model.

Errors

BirdNET writes its error log to `~/birdnet_error_log.txt`. Read it when a run fails without a clear message in the job log. `BUILD_INFO.txt` in each installation directory records the patches applied to the upstream package on Hydra.

Last updated 2026-10-03

BLAST

BLAST matches query nucleotide or protein sequences against a database. The module is `bio/blast`. Loading it sets `BLASTDB` to the local copy of the NCBI databases under `/scratch/dbs/blast/v5`, so `-db nt` needs no path. The copies (`nr`, `nt`, `refseq_protein`, `swissprot`, `taxdb`, `mito`, `core_nt`) are listed under [Local databases](#). NCBI's manual at <https://www.ncbi.nlm.nih.gov/books/NBK279675/> describes every option.

Search a database

Use `blastn` for nucleotide queries and `blastp` for protein. The options are the same.

Option	Meaning
<code>-task</code>	the search type, for example <code>megablast</code>
<code>-db</code>	the database, for example <code>nt</code> or <code>nr</code>
<code>-query</code>	the input FASTA file
<code>-outfmt</code>	the output format; <code>5</code> is XML, which Blast2GO and other tools read
<code>-out</code>	the output file
<code>-num_threads</code>	threads; set it to <code>\$NSLOTS</code> in a job that requested <code>-pe mthread N</code>

blast.job

```
#$ -S /bin/sh
#$ -N blast -cwd -j y -o blast.log
#$ -q lThM.q
#$ -l mres=32G,h_data=32G,h_vmem=32G,himem
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
module load bio/blast
blastn -task megablast -db nt -query queries.fa -outfmt 5 -out queries.xml
gzip queries.xml
echo = `date` job $JOB_NAME done
```

BLAST output is large and repetitive. Compress it in the job, as above. Remove duplicate sequences from the input first. A query file in which most sequences hit uses memory in proportion to the hits and can exceed even a high-memory node. Split such files, as shown below.

Build a database

`makeblastdb` builds a database from a FASTA file. Give it `-dbtype nucl` for nucleotide or `-dbtype prot` for protein.

makedb.job

```

## -S /bin/sh
## -N makedb -cwd -j y -o makedb.log
## -q sThM.q
## -l mres=12G,h_data=12G,h_vmem=12G,himem
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
module load bio/blast
makeblastdb -in sequences.fa -parse_seqids -dbtype nucl
echo = `date` job $JOB_NAME done

```

Split a large search across jobs

`-num_threads` parallelises one search on one node. For a large input, splitting the FASTA into files of 1,000 to 10,000 sequences and searching each in its own job finishes sooner and uses less memory per job. Concatenate the results afterwards. A [job array](#) does this from one job file. The loop below does it with one job per file.

blast-part.job

```

## -S /bin/sh
## -cwd -j y
## -q mThC.q
## -l mres=4G,h_data=4G,h_vmem=4G
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
module load bio/blast
blastp -db nr -query $1 -outfmt 5 -out $1.out
echo = `date` job $JOB_NAME done

```

```
$ for x in *.fa; do qsub -N blast-$x -o $x.log blast-part.job $x; done
```

`$1` is the file name passed after the job file on the `qsub` line. See [Pass arguments to the job](#).

Last updated 2026-10-03

Blast2GO

Hydra has a license for [Blast2GO Command Line \(manual\)](#) with a local copy of the mapping database, so mapping and annotation run without the network round trips of the desktop version. The module is `bio/blast2go`.

License

The license allows one execution of the command-line program at a time, on one node. Jobs run in the queue `1Tb2g.q`, which keeps one slot on that node free for Blast2GO. A second Blast2GO job waits until the first finishes.

Use Blast2GO for mapping and annotation only

Because one job runs at a time, do not run the BLAST step inside Blast2GO. Run BLAST separately (see [BLAST](#)), split across many jobs if the input is large, with `-outfmt 5`. Blast2GO reads the XML with `-loadblast31` or `-loadblast`.

Job file

Request the queue and the `b2g` resource. The queue allows 36 GB of memory and one slot per job, with no parallel environment, and has the time limits of the long queues (30 days of CPU, 60 days elapsed). The module defines two commands. `runblast2go` starts Blast2GO with the Java options set. `hydracliprop` copies a `cli.prop` template configured for the local database into the current directory. It overwrites an existing one, so run it once.

`b2g.job`

```

## -S /bin/sh
## -N b2g -cwd -j y -o b2g.log
## -q lTb2g.q
## -l b2g,mres=32G,h_data=32G,h_vmem=32G
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
module load bio/blast2go
hydracliprop
runblast2go \
  -properties cli.prop \
  -loadfasta /share/apps/bioinformatics/blast2go/1.5.1/example_data/
plant_nucleotide.fasta \
  -loadblast31 /share/apps/bioinformatics/blast2go/1.5.1/example_data/
plant_blast_31.zip \
  -useobo $BLAST2GO_OBO \
  -mapping \
  -annotation \
  -savebox example.box \
  -savereport example.pdf
echo = `date` job $JOB_NAME done

```

Variable	Default	Effect
BLAST2GO_HEAP_SIZE	1024m	Java maximum heap for runblast2go
BLAST2GO_TEMP	the directory you are in when the module loads	-tempfolder, where logs and temporary files go; in a job with -cwd that is the job's directory
BLAST2GO_OBO	set by the module	the OBO file matching the local mapping database, from biobam

The mapping database is large and we keep only the current version. An update removes the previous one.

Graphs and statistics

-statistics all writes every available statistic as PNG images, CSV and .b2g files. The manual lists the individual statistics and the report options.

Last updated 2026-10-03

BUSCO

BUSCO (Benchmarking Universal Single-Copy Orthologs) assesses the completeness of a genome, transcriptome or protein set by searching for universal and lineage-specific single-copy orthologs from [OrthoDB](#). The module is `bio/busco`. `module -t avail 2>&1 | grep busco` lists the versions.

Augustus

BUSCO can run Augustus for gene prediction, which needs a writable copy of the Augustus `config` directory. Loading the module defines the alias `cp_aug_config`, which copies that directory into the current directory, and sets `AUGUSTUS_CONFIG_PATH` to `config` under the current directory. To keep one copy elsewhere, set the variable in the job file:

```
export AUGUSTUS_CONFIG_PATH=/scratch/genomics/USERNAME/augustus/config
```

Job file

This job runs BUSCO on a genome assembly with 24 threads in a high-memory queue:

`busco.job`

```
## -S /bin/sh
## -N busco -cwd -j y -o busco.log
## -q mThM.q
## -pe mthread 24
## -l mres=240G,h_data=10G,h_vmem=10G,himem
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
echo + NSLOTS = $NSLOTS
module load bio/busco/6.0.0
busco -i assembly.fa.gz -m genome -l vertebrata --cpu $NSLOTS
echo = `date` job $JOB_NAME done
```

Last updated 2026-10-03

MaSuRCA

MaSuRCA assembles genomes by building super-reads from Illumina data and assembling those, with or without long reads (PacBio, Nanopore) for a hybrid assembly. The module is `bio/masurca`. The MaSuRCA manual covers the parameters. What follows is how to run it on Hydra.

MaSuRCA runs in two jobs. A short one turns the configuration file into an `assemble.sh` script, and a long one runs the script.

1. Put the reads under `/scratch` (see [Data transfer](#)), as FASTQ for Illumina and FASTA or FASTQ for long reads.
2. Generate a configuration file and edit it:

```
$ module load bio/masurca
$ masurca -g masurca.config
```

In the `DATA` section, give each library a unique two-letter prefix, the fragment mean and standard deviation, and the full paths of the read files. In `PARAMETERS`, keep `USE_GRID=0` (MaSuRCA's grid mode is not set up on Hydra) and set `NUM_THREADS` to the slot count you will request for the second job. The comments in the file explain the rest.

```
DATA
PE= pe 500 50 /scratch/genomics/USERNAME/reads/frag_1.fastq /scratch/genomics/
    USERNAME/reads/frag_2.fastq
JUMP= sh 3600 200 /scratch/genomics/USERNAME/reads/short_1.fastq /scratch/
    genomics/USERNAME/reads/short_2.fastq
#NANOPORE=/scratch/genomics/USERNAME/reads/nanopore.fa
END
```

3. Run the first job, which writes `assemble.sh` in seconds:

`masurca-config.job`

```
## -S /bin/sh
## -N masurca-config -cwd -j y -o masurca-config.log
## -q sThC.q
## -l mres=2G,h_data=2G,h_vmem=2G
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
module load bio/masurca
masurca masurca.config
echo = `date` job $JOB_NAME done
```

4. Run the assembly. Memory depends on the genome size. The high-memory queues allow up to 450 GB per slot, and `uTxLM.rq` more, by request to SI-HPC@si.edu. This job runs for hours to days.

masurca-assemble.job

```
#$ -S /bin/sh
#$ -N masurca-assemble -cwd -j y -o masurca-assemble.log
#$ -q lThM.q
#$ -pe mthread 16
#$ -l mres=480G,h_data=30G,h_vmem=30G,himem
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
echo + NSLOTS = $NSLOTS
module load bio/masurca
./assemble.sh
echo = `date` job $JOB_NAME done
```

Last updated 2026-10-03

SMC++

SMC++ infers population size history from whole-genome sequences. The module `bio/smc++` provides release 1.15.2. The developers publish later releases only as a [Docker image](#), which runs on Hydra under Singularity (see [Containers](#)).

Run the current version from the container

1. Build the `.sif` file from the Docker image, once, on a login node:

```
$ singularity pull smcpp.sif docker://terhorst/smcpp:latest
$ ls -l smcpp.sif
-rwxrwxr-x 1 USERNAME USERNAME 257298432 Sep 20 13:09 smcpp.sif
```

2. Check it runs:

```
$ singularity run -C smcpp.sif vcf2smc
usage: smc++ vcf2smc [-h] [-v] [--cores CORES] [-d sample_id sample_id]
...
smc++ vcf2smc: error: the following arguments are required: vcf.gz, out[.gz],
contig, pop1
```

3. Run a command with the current directory bound to `/mnt` in the container. Only directories you bind are visible inside it:

```
singularity run -C --bind $PWD:/mnt smcpp.sif vcf2smc /mnt/vcf.gz /mnt/
chr1.smc.gz chr1 CEU:NA12878,NA12879
```

This is the container form of `smc++ vcf2smc vcf.gz chr1.smc.gz chr1 CEU:NA12878,NA12879` with `vcf.gz` in the current directory.

Put step 3 in a job file for anything longer than a test. A `singularity run` line needs no module.

Last updated 2026-09-30

SPAdes

The [SPAdes](#) assembler handles microbial and organelle genomes and target-enrichment data such as UCE and exon capture. The module is `bio/spades`.

Memory

Give `spades.py` the memory the job reserved, in GB, with `-m`. A job with `-l mres=96G` runs `spades.py -m 96`. Without `-m`, SPAdes assumes 250 GB and the scheduler kills it when it exceeds the job's limit.

Temporary files

`spades.py` creates many temporary files, over a million with the `--careful` option, whose mismatch-correction stage writes a separate file for each contig. On `/scratch` this is slow and counts against your file quota. Put them on the node's local SSD instead. Request the space with `-l ssd_res=SIZE`, load `tools/ssd`, and pass `--tmp-dir $SSD_DIR` (see [Use a node's local SSD](#)). `-v SSD_SAVE_MAX=0` tells the scheduler not to save anything left on the SSD when the job ends, because the temporary files are not needed.

`spades.job`

```
#$ -S /bin/sh
#$ -N spades -cwd -j y -o spades.log
#$ -q lThM.q
#$ -pe mthread 8
#$ -l mres=96G,h_data=12G,h_vmem=12G,himem
#$ -l ssd_res=200G -v SSD_SAVE_MAX=0
#
echo + `date` job $JOB_NAME started in $QUEUE with jobID=$JOB_ID on $HOSTNAME
echo + NSLOTS = $NSLOTS
module load bio/spades
module load tools/ssd
spades.py \
  -o sample \
  --pe1-1 sample_R1_PE_trimmed.fastq.gz \
  --pe1-2 sample_R2_PE_trimmed.fastq.gz \
  --tmp-dir $SSD_DIR \
  -m 96 \
  -t $NSLOTS
echo = `date` job $JOB_NAME done
```

SPAdes under Phyluce

[Phyluce](#)'s `phyluce_assembly_assemblo_spades` calls `spades.py` itself and has no option for `--tmp-dir`. Give it a wrapper that adds the option when the job has an SSD:

`spades-tmp.sh`

```
#!/bin/sh
if [ -n "$SSD_DIR" ]; then
    SPADES_TMP="--tmp-dir $SSD_DIR"
else
    SPADES_TMP=""
fi
spades.py $SPADES_TMP "$@"
```

Make it executable and name it in `~/.phyluce.conf`:

`~/.phyluce.conf`

```
[binaries]
spades:/scratch/genomics/USERNAME/bin/spades-tmp.sh
```

Last updated 2026-09-30

Local bioinformatics databases

Hydra keeps large, widely used reference databases under `/scratch/dbs`, so that each user does not download a copy. Email SI-HPC@si.edu to request a database.

Old versions are not kept

An update removes the previous copy. If your pipeline depends on a particular version, record the version from the `README` in the database directory.

`/scratch/dbs/blast`

This directory holds the databases for [BLAST](#) and [diamond](#).

```
/scratch/dbs/blast
├─ v5/                # v5 of the nr, nt, refseq_protein, swissprot, taxdb databases
│   └─ README         # download commands, database and version information
│   └─ mito/          # v5 of mito
└─ core_nt            # v5 of core_nt
```

Loading `bio/blast` sets `BLASTDB` to `/scratch/dbs/blast/v5`, so `-db nt` finds the local copy. See [BLAST](#).

`/scratch/dbs/dfam`

```
/scratch/dbs/dfam
├─ README             # database and version information
├─ 3.8/               # Dfam FamDB 3.8
└─ 3.9/               # Dfam FamDB 3.9
```

Local copies of the [Dfam FamDB](#) transposable-element partitions, used by the RepeatMasker modules: Dfam 3.8 for RepeatMasker 4.1.x, from https://www.dfam.org/releases/Dfam_3.8/families/FamDB/, and Dfam 3.9 for RepeatMasker 4.2.x, from https://www.dfam.org/releases/Dfam_3.9/families/FamDB/. `README.txt` in each directory lists the taxonomic grouping of the partitions. In the RepeatMasker installations under `/share/apps/bioinformatics/repeatmasker`, these `.h5` files are linked into `share/RepeatMasker/Libraries/famdb`.

`/scratch/dbs/kraken`

```
/scratch/dbs/kraken
├─ README             # database and version information
├─ core_nt/           # "very large collection, inclusive of GenBank, RefSeq, TPA and PDB"
└─ standard/          # "RefSeq archaea, bacteria, viral, plasmid, human, UniVec_Core"
```

Local copies of the `standard` and `core_nt` Kraken 2 indexes from <https://benlangmead.github.io/aws-indexes/k2>.

```
/scratch/dbs/kraken-uniq
```

```
/scratch/dbs/kraken-uniq
├─ README          # database and version information
└─ standard/       # "archaea, bacteria, viral, human, UniVec_Core"
```

A local copy of the `standard` `KrakenUniq` index from <https://benlangmead.github.io/aws-indexes/k2>.

Last updated 2026-10-04

Data Transfer

There are three ways to move files to and from Hydra. Which one to use depends on how much data you are moving and where it is.

Method	Use it for	Page
<code>scp</code> , <code>sftp</code> , <code>rsync</code> and graphical clients (WinSCP, FileZilla)	files between your computer and Hydra, up to a few tens of gigabytes	scp, sftp and rsync
Globus	large transfers, transfers between institutions, and transfers that should keep going after you close your laptop	Globus
rclone	cloud storage: Dropbox, Google Drive, S3	rclone and cloud storage

A transfer to Hydra starts from a computer on the Smithsonian network or the SI VPN. The login nodes do not accept connections from elsewhere. A transfer from Hydra can go to any host the login nodes can reach.

Copy data to `/scratch` or `/data`, not to `/home`

The home directory has a quota of 350 GB and 9 million files, and it is where `scp` puts files when you give no destination. We ask that every transfer name a destination under `/scratch/genomics/USERNAME` or `/data`. Do not use `/tmp`. It is small and shared.

`/store` is not mounted on the compute nodes. Copies between `/store` and `/scratch` or `/data` run as I/O jobs. See [Use /store and the I/O queue](#).

Further reading

- [Filesystems](#) for which partition to put data on
- [Check your disk usage and quotas](#)
- [Logging in and passwords](#) for the login-node names and the VPN

Last updated 2026-10-03

scp, sftp and rsync

scp, **sftp** and **rsync** are the standard Unix tools for copying files over an ssh connection. **scp** copies files with a single command. **sftp** opens a session in which you move around and copy files one at a time. **rsync** copies whole directories and, on a second run, only what has changed. Hydra's login nodes accept all three from any computer on the Smithsonian network or the SI VPN. Graphical clients such as WinSCP and FileZilla use the same protocol and work the same way. From Hydra outward, the same tools reach any host that accepts an ssh connection from the login nodes.

The three tools are installed by default on macOS and Linux. Windows 10 (version 1803 and later) and Windows 11 include **scp** and **sftp**. **rsync** is not part of Windows. Use it through the Windows Subsystem for Linux (WSL), or use a graphical client instead.

Use rsync for anything over about 70 GB

rsync can resume an interrupted copy and limit its own speed. For large transfers we ask you to cap the rate at 20 MB/s (about 70 GB per hour) with `--bwlimit=20000`, so that one transfer does not fill the link for everyone. If that limit is a problem for your work, email SI-HPC@si.edu.

Give every transfer a destination under `/scratch` or `/data`. The home directory has a small quota, and `/tmp` is small and shared.

Copy files with scp

Open a terminal. On macOS that is Terminal, in `/Applications/Utilities`. On Windows it is Command Prompt or PowerShell. Use `cd` to go to the directory the files are in.

Copy one file to a directory on Hydra:

```
$ scp -p results.tar USERNAME@hydra-login01.si.edu:/scratch/genomics/USERNAME/project/
```

Copy several files at once, with a wildcard:

```
$ scp -p reads_*.fastq.gz USERNAME@hydra-login01.si.edu:/scratch/genomics/USERNAME/project/
```

Copy a file from Hydra to the current directory on your computer:

```
$ scp -p USERNAME@hydra-login01.si.edu:/scratch/genomics/USERNAME/project/results.tar .
```

Copy several files from Hydra with a wildcard. The quotes make Hydra expand the wildcard rather than your own shell.

```
$ scp -p 'USERNAME@hydra-login01.si.edu:/scratch/genomics/USERNAME/project/*.log' .
```

The destination directory must exist. `-p` keeps the files' modification times, which is useful when you later want to know which copy is newer. `man scp` lists the other options.

Copy or synchronise with rsync

`rsync` copies only the files that are new or have changed since the last run. A second run after an interruption finishes the job rather than starting over, and a run over a directory you have already copied moves only what you changed.

Copy a directory to Hydra, at a limited rate:

```
$ rsync -avz --bwlimit=20000 project/ USERNAME@hydra-login01.si.edu:/scratch/
genomics/USERNAME/project/
```

Copy results back from Hydra:

```
$ rsync -avz USERNAME@hydra-login01.si.edu:/scratch/genomics/USERNAME/project/
results/ results/
```

See what a run would copy without copying anything:

```
$ rsync -avzn project/ USERNAME@hydra-login01.si.edu:/scratch/genomics/USERNAME/
project/
```

The options mean the following. `-a` copies directories recursively and keeps permissions and times. `-v` lists each file as it goes. `-z` compresses in transit. `-n` is the dry run. A trailing slash on the source copies the directory's contents. Without it, `rsync` copies the directory itself into the destination. `man rsync` covers the rest.

Use sftp

`sftp` opens a session on Hydra and takes commands, which suits a few files here and there.

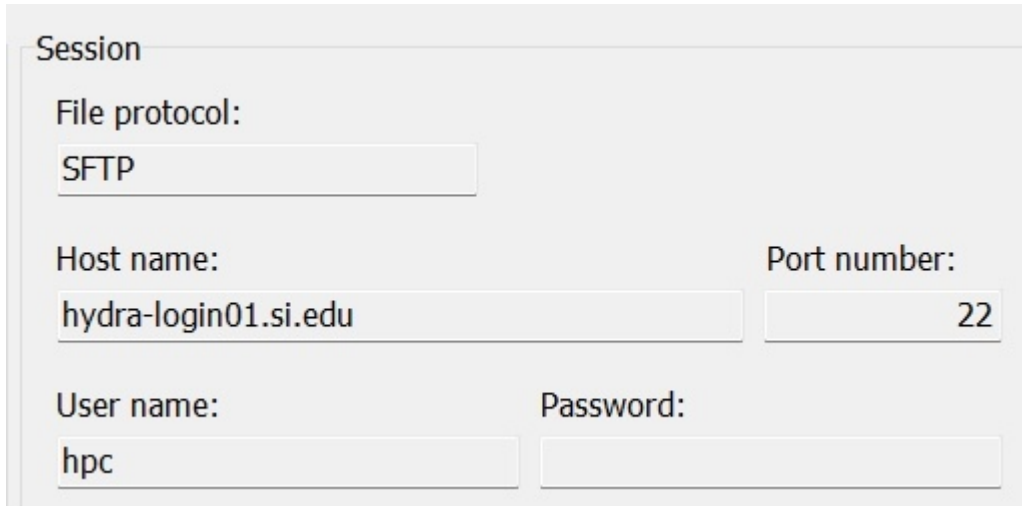
```
$ sftp USERNAME@hydra-login01.si.edu
sftp> cd /scratch/genomics/USERNAME/project
sftp> put results.tar
sftp> get summary.csv
sftp> exit
```

You will use four commands most. `cd` changes the directory on Hydra. `lcd` changes the directory on your computer. `put` uploads a file and `get` downloads one. `man sftp` lists the rest.

Use WinSCP on Windows

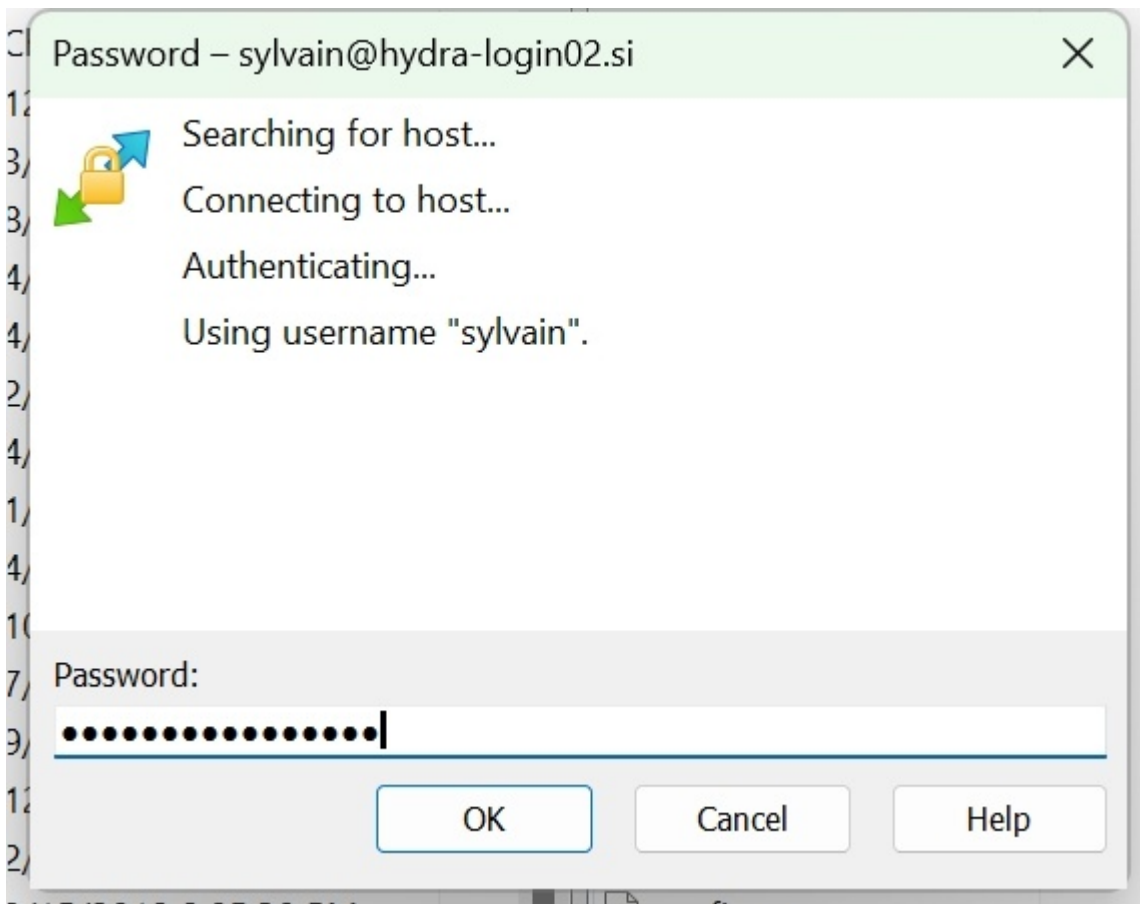
WinSCP is a free Windows program that copies files by drag and drop over sftp.

1. In the Login window, set the file protocol to SFTP, the host name to `hydra-login01.si.edu`, the port to 22, and your Hydra username. Leave the password blank and click **Login**:



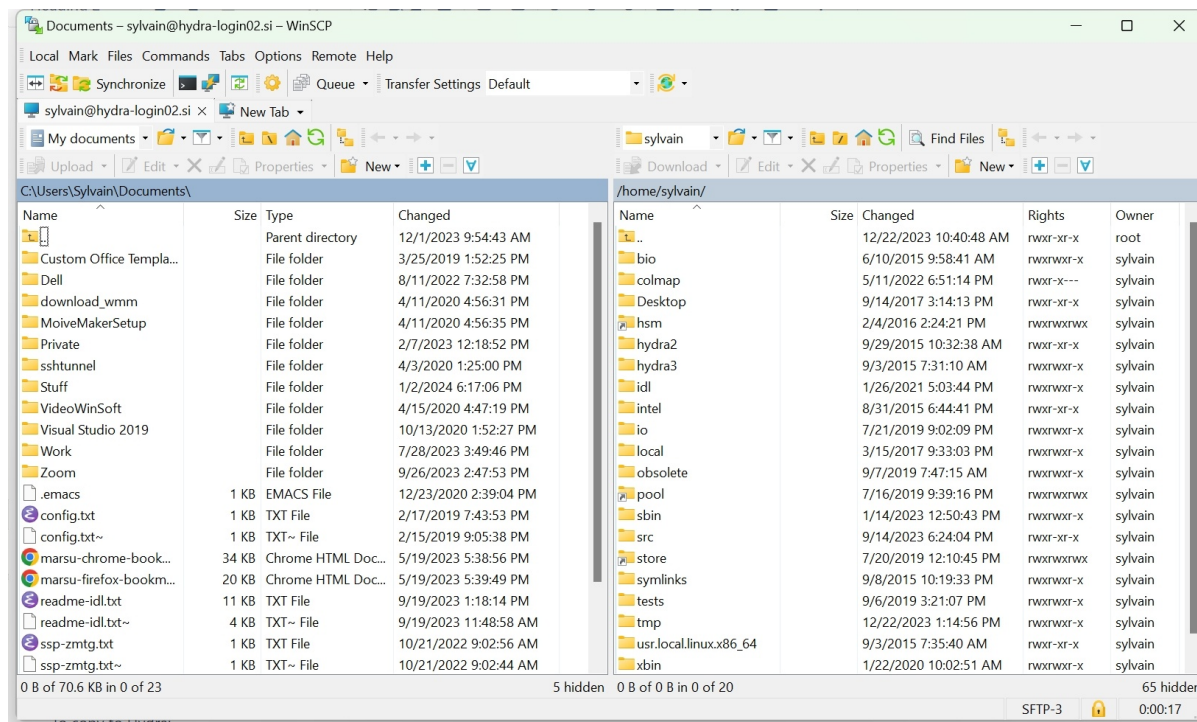
The image shows the 'Session' dialog box in WinSCP. It has four input fields: 'File protocol:' with 'SFTP' selected, 'Host name:' with 'hydra-login01.si.edu', 'Port number:' with '22', and 'User name:' with 'hpc'. The 'Password:' field is empty. The 'Login' button is not visible in this crop.

2. Accept the host key the first time. Then enter your Hydra password when the login dialog asks for it:

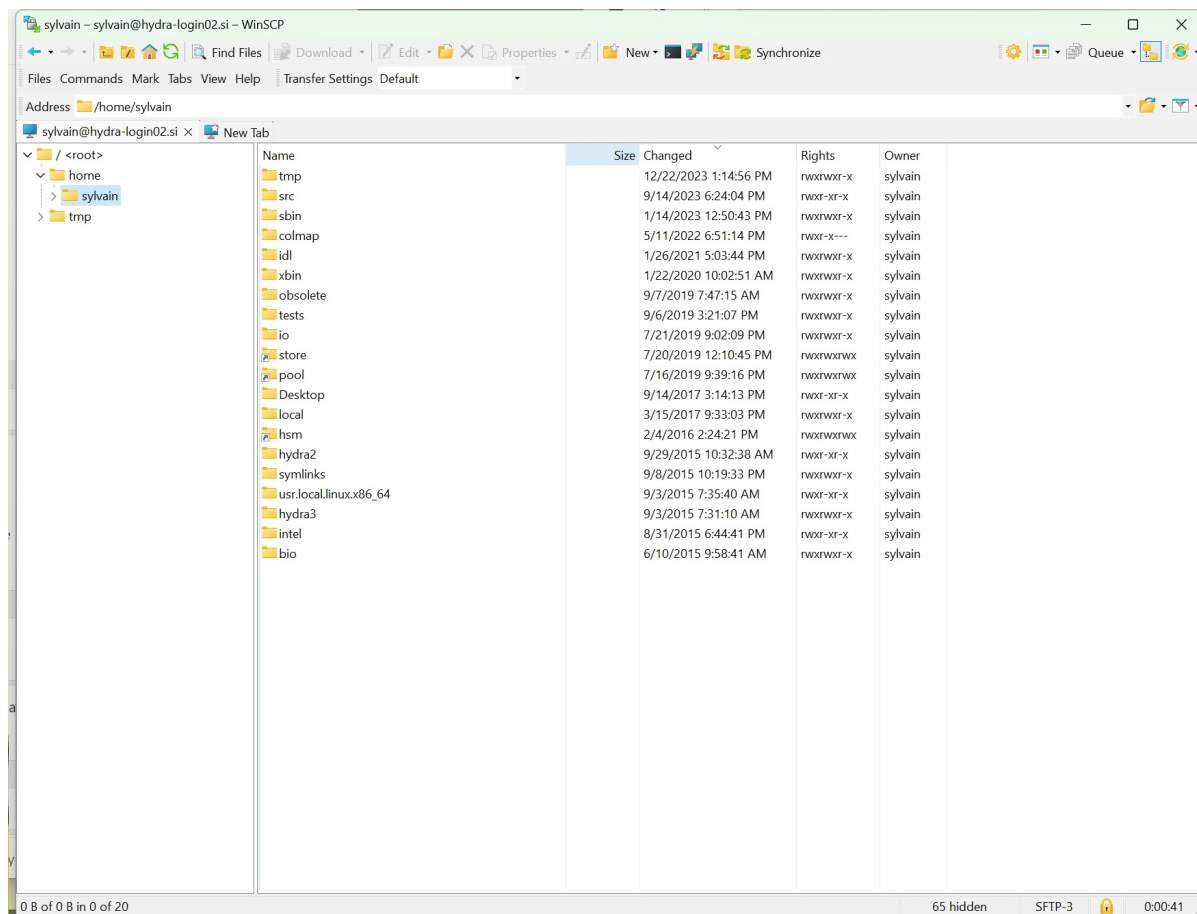


The image shows a 'Password' dialog box titled 'Password – sylvain@hydra-login02.si'. It contains a list of status messages: 'Searching for host...', 'Connecting to host...', 'Authenticating...', and 'Using username "sylvain"'. Below the messages is a 'Password:' label and a text box filled with dots. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

3. Your computer is in the left pane and your Hydra home directory in the right. Navigate the right pane to a directory under `/scratch` or `/data` before copying anything:



4. Drag files between the panes to copy them. The Transfer Settings bar shows progress:

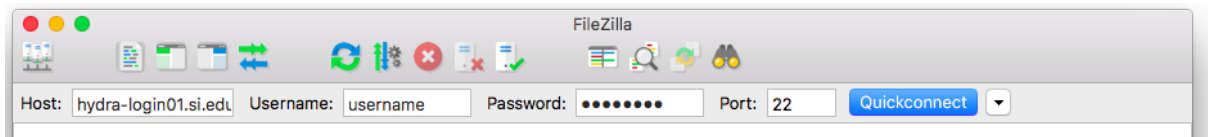


WinSCP can also show only the Hydra side ("Explorer" mode), in which case you drag files from Windows File Explorer. Its Preferences say which.

Use FileZilla on any system

[FileZilla](#) is a free client for macOS, Windows and Linux. The screenshots are from a Mac. The other systems look the same.

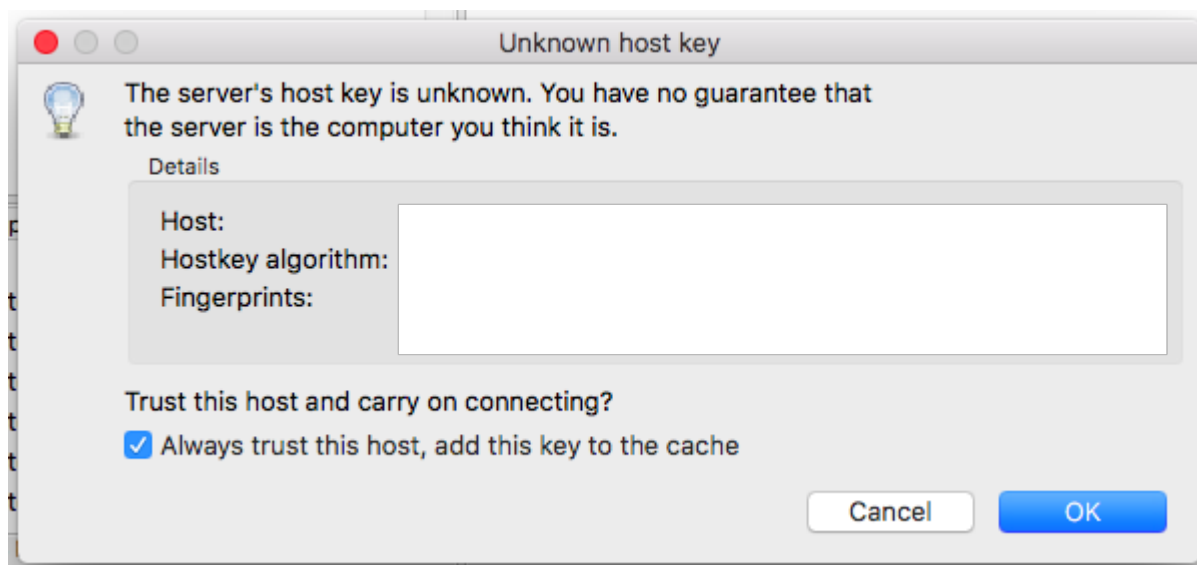
1. In the Quickconnect bar, enter the host `hydra-login01.si.edu`, your Hydra username, your password and port 22, then click **Quickconnect**:



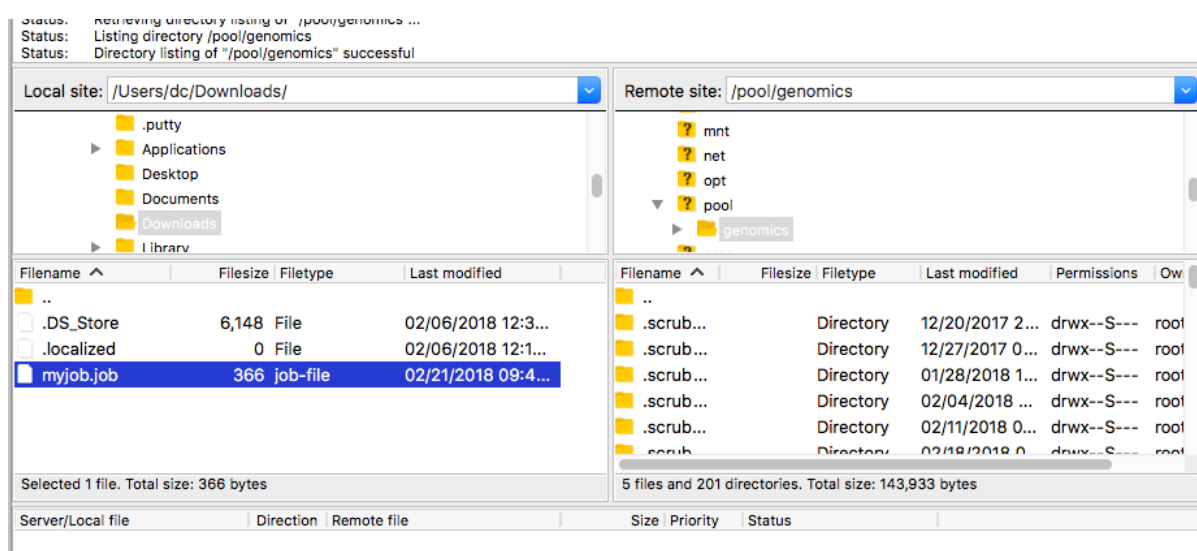
2. The first time, FileZilla asks whether to save passwords. Choose **Do not save passwords**:



3. Accept the host key when FileZilla shows it. Tick **Always trust this host** so it does not ask again:



4. Your computer is on the left and Hydra on the right. Navigate the right side to a directory under `/scratch` or `/data`, or type the path into the **Remote site** box, then drag files between the two sides:



We do not recommend Cyberduck. It keeps a process busy on the login node, and the login-node limits then kill it.

Keep the load down

`rm`, `mv` and `cp` on thousands of files load the file servers as much as a transfer does. Run large file operations one after another rather than several at once. Anything that takes more than a few minutes on a login node should run under `qsh` or as a job instead. See [Start an interactive session](#).

Further reading

- [Filesystems](#) for the partitions and their quotas
- [Globus](#) for transfers too large or too long for `rsync`

- [Logging in and passwords](#) for ssh keys, which let `scp` and `rsync` run without a password

Last updated 2026-10-03

Globus

Globus is a service for moving data between storage systems. You set a transfer up in a web browser, and Globus runs it for you. It retries after network failures, checks that every file arrived intact, and emails you when it is done. Nothing has to stay logged in while it runs. We recommend it for transfers of hundreds of gigabytes, for data at another institution, and for any transfer that has to survive a closed laptop.

More in-depth instructions for using Globus beyond Hydra are available at <https://smithsonian.github.io/globus-docs/>. Here you sign in with your SI account, find the Hydra collections, and run a transfer.

Four terms come up in every step.

Term	Meaning
Collection	A storage system that Globus can reach, with a name you search for. Hydra's are <code>SI_hydra_scratch</code> and the others listed below.
Endpoint	The computer that serves a collection. Hydra's endpoints are <code>hydra-globus01</code> and <code>hydra-globus02</code> . You never address them directly.
Globus Connect Personal	A program that turns your own computer into a collection, so you can transfer to and from it.
Transfer	A copy between two collections that Globus runs for you.

Sign in

1. Open app.globus.org and choose **Smithsonian Institution** as your organization:



Log in to use Globus Web App

Use your existing organizational login

e.g., university, national lab, facility, project

Smithsonian Institution

By selecting Continue, you agree to Globus [terms of service](#) and [privacy policy](#).

Continue



Globus uses CILogon to enable you to Log In from this organization. By clicking Continue, you agree to the [CILogon privacy policy](#) and you agree to share your username, email address, and affiliation with CILogon and Globus. You also agree for CILogon to issue a certificate that allows Globus to act on your behalf.

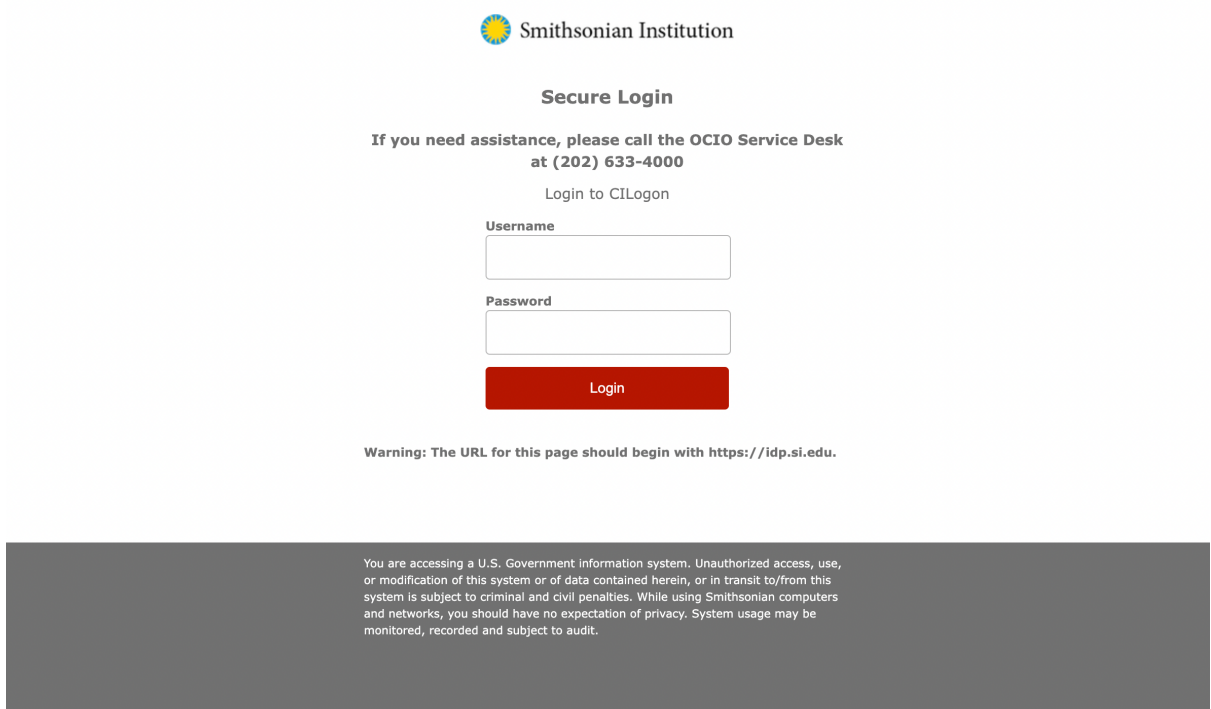
OR

 Sign in with Google

 Sign in with ORCID iD

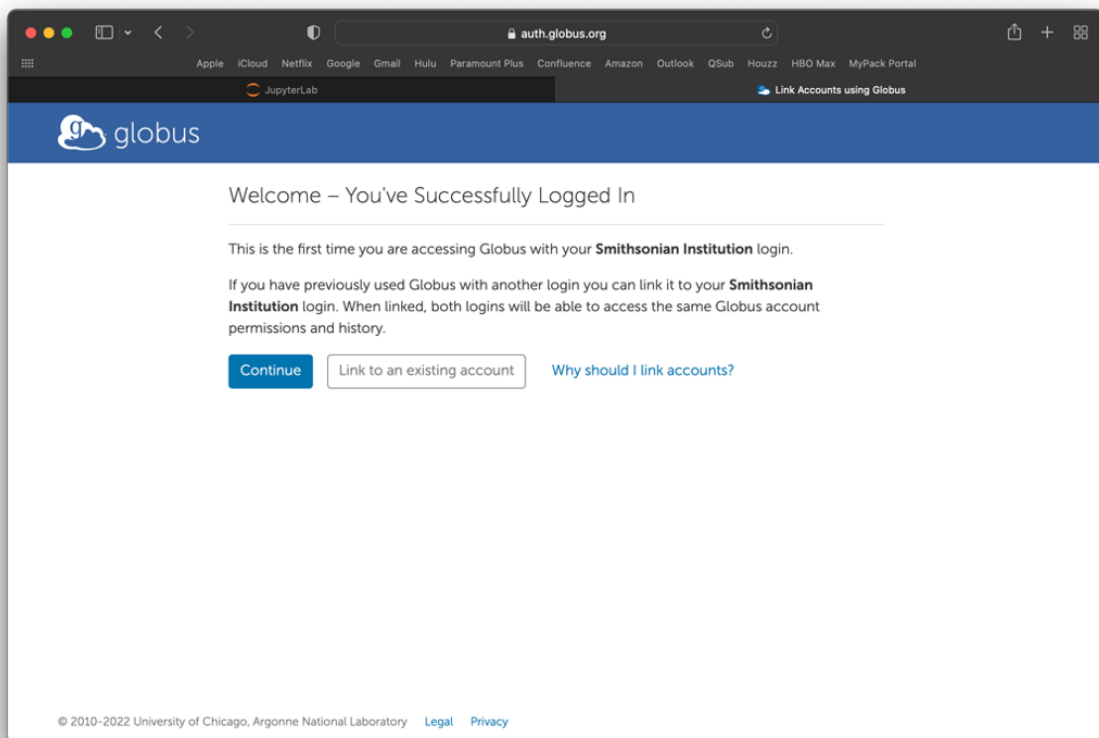
Didn't find your organization? Then use [Globus ID to sign in](#). ([What's this?](#))

2. Sign in on the Smithsonian page with your SI network username and password. The username is the part of your email address before @si.edu. This is your SI account. It is separate from your Hydra account:



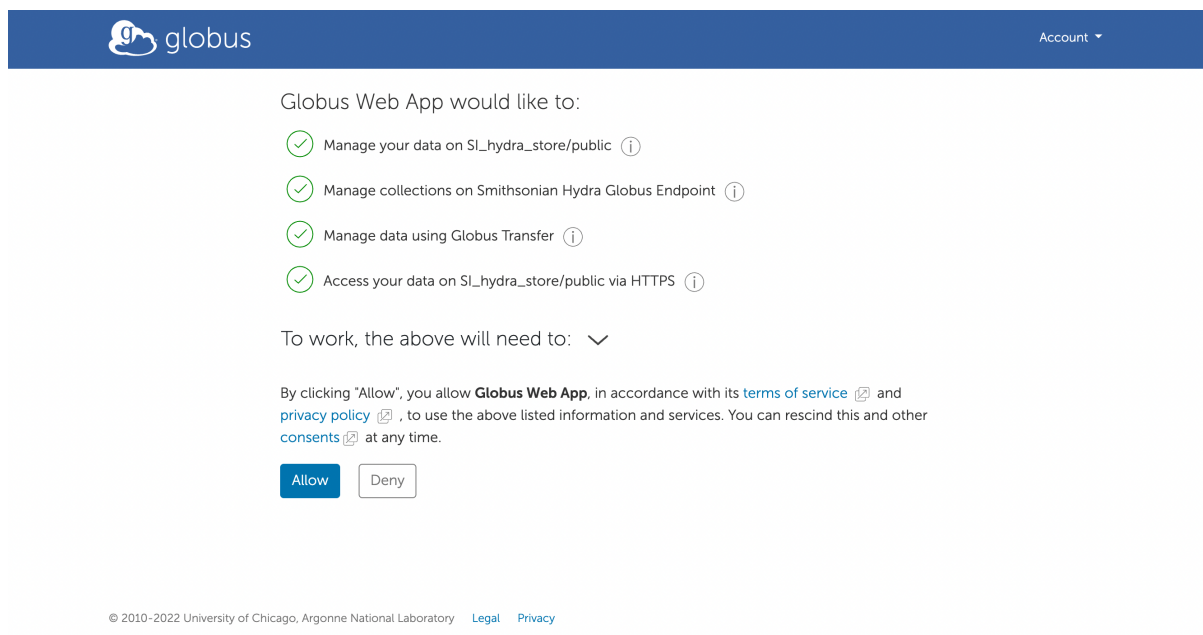
The image shows the Smithsonian Institution's secure login page. At the top is the Smithsonian logo and the text "Smithsonian Institution". Below this is the heading "Secure Login". A message states: "If you need assistance, please call the OCIO Service Desk at (202) 633-4000". Below this is the text "Login to CILogon". There are two input fields: "Username" and "Password". Below the password field is a red "Login" button. At the bottom, a warning message reads: "Warning: The URL for this page should begin with https://idp.si.edu." Below the login form is a large grey box containing a disclaimer: "You are accessing a U.S. Government information system. Unauthorized access, use, or modification of this system or of data contained herein, or in transit to/from this system is subject to criminal and civil penalties. While using Smithsonian computers and networks, you should have no expectation of privacy. System usage may be monitored, recorded and subject to audit."

3. The first time, Globus asks whether you have an existing Globus account to link. If you do not, choose **Continue** and complete the sign-up:



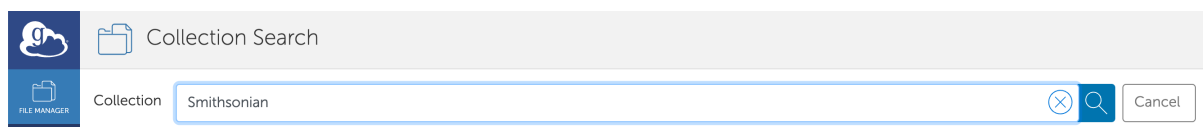
The image shows a web browser window displaying the Globus authentication page at auth.globus.org. The browser's address bar shows the URL. The page has a blue header with the Globus logo and the text "Link Accounts using Globus". The main content area says "Welcome – You've Successfully Logged In". Below this, it states: "This is the first time you are accessing Globus with your Smithsonian Institution login." It then explains: "If you have previously used Globus with another login you can link it to your Smithsonian Institution login. When linked, both logins will be able to access the same Globus account permissions and history." There are three buttons: a blue "Continue" button, a white "Link to an existing account" button, and a blue link "Why should I link accounts?". At the bottom, there is a copyright notice: "© 2010-2022 University of Chicago, Argonne National Laboratory" and links for "Legal" and "Privacy".

4. Allow the Globus web app to manage data on your behalf when it asks:



Open your Hydra directory

1. In the File Manager, type **Smithsonian** in the **Collection** search box:

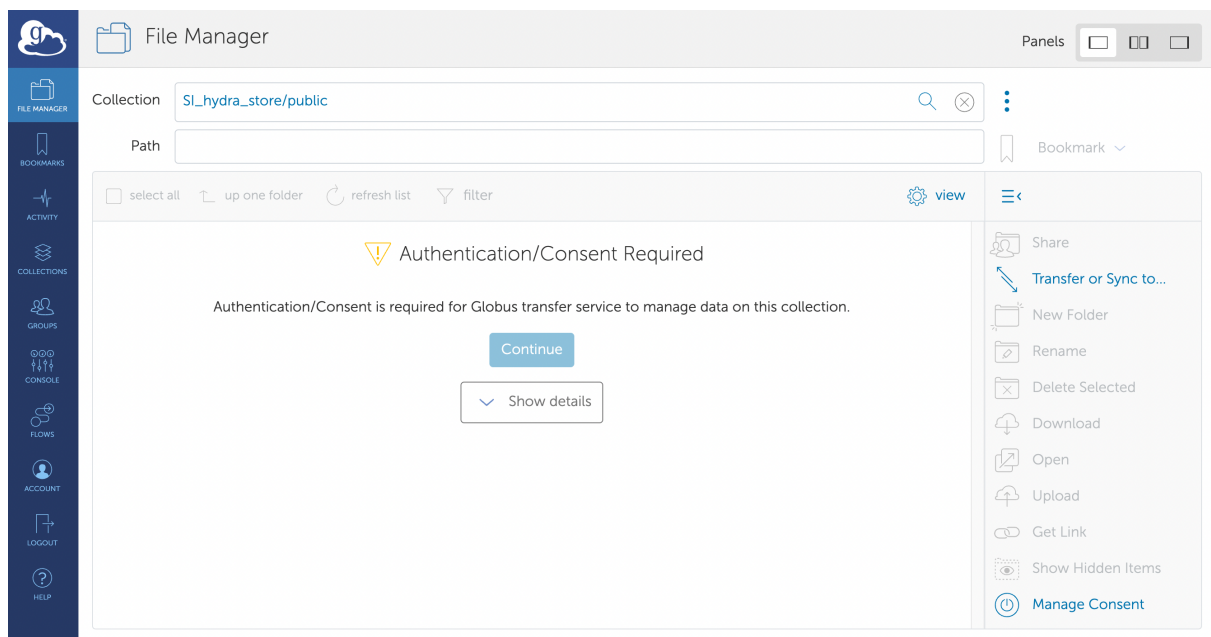


The Hydra collections begin with **SI_hydra_**:

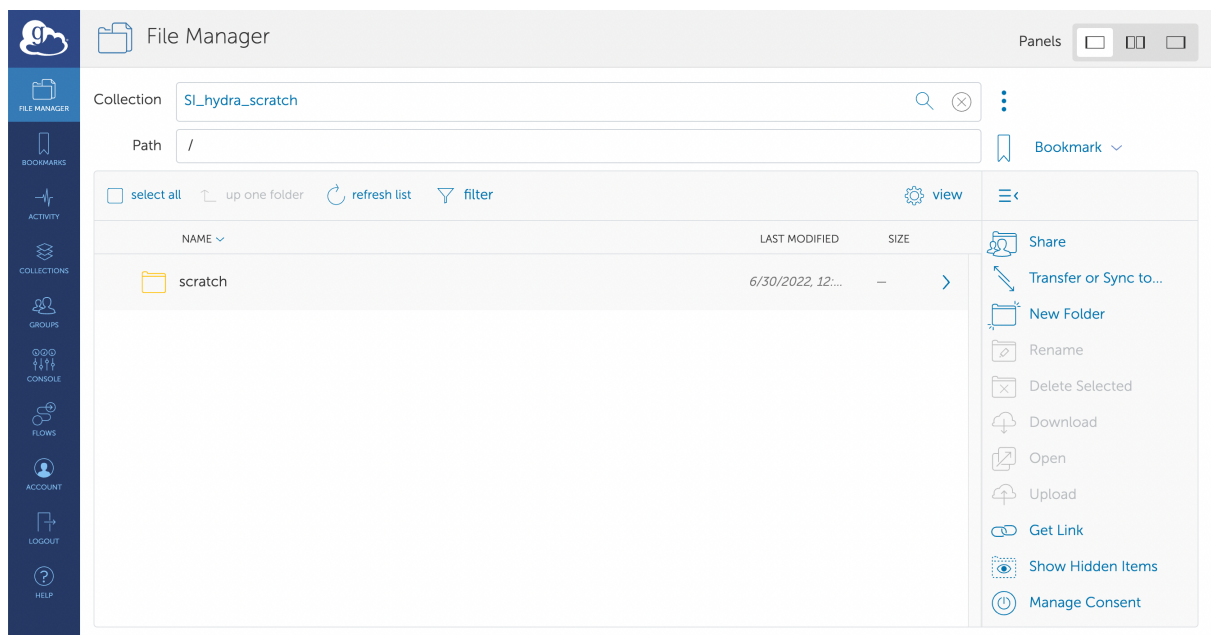
 SI_hydra_store/public Owner: whiteae@si.edu Domain: m-660a5e.9d21.8443.data.globus.org Description: Smithsonian Hydra /store/public access for SI users	⋮
 SI_hydra_scratch_SAO Owner: whiteae@si.edu Domain: m-6905a2.9d21.8443.data.globus.org Description: Smithsonian Hydra /scratch access for SAO users	⋮
 SI_hydra_scratch Owner: whiteae@si.edu Domain: m-066c43.9d21.8443.data.globus.org Description: Collection of GPFS directories for hydra.si.edu genomics users	⋮

Collection	Path on Hydra
SI_hydra_scratch	/scratch
SI_hydra_scratch_SAO	/scratch , the SAO project partitions
SI_hydra_store/public	/store/public

2. The first time you open a collection, Globus asks you to allow access to it:



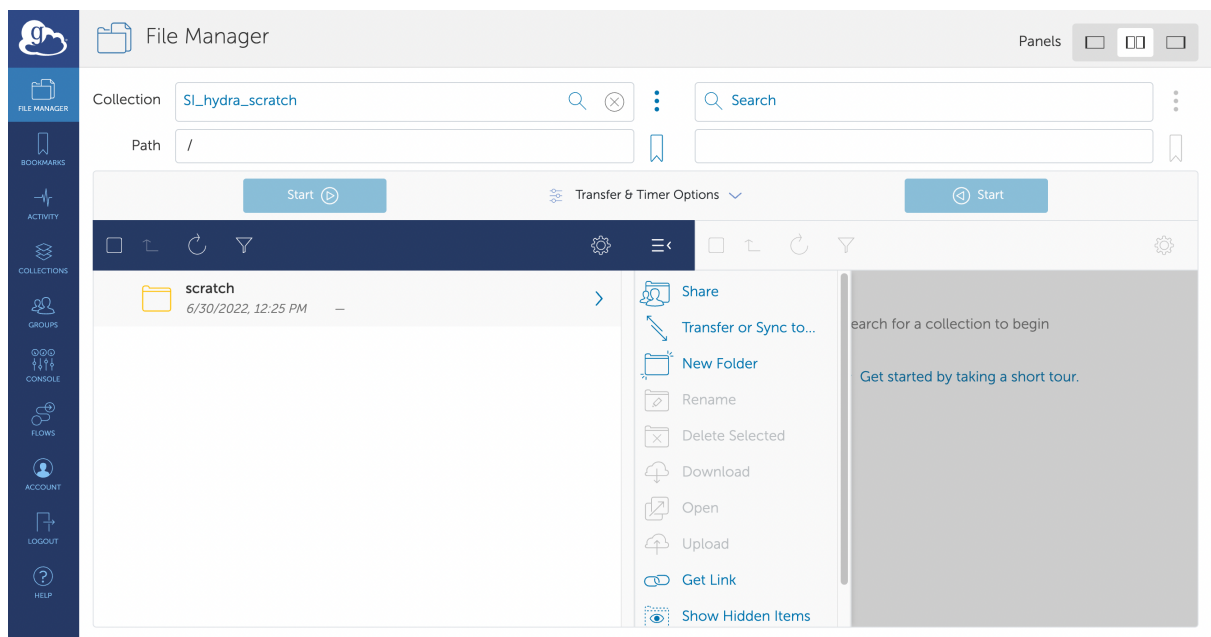
3. Navigate to your directory, for example `/scratch/genomics/USERNAME` :



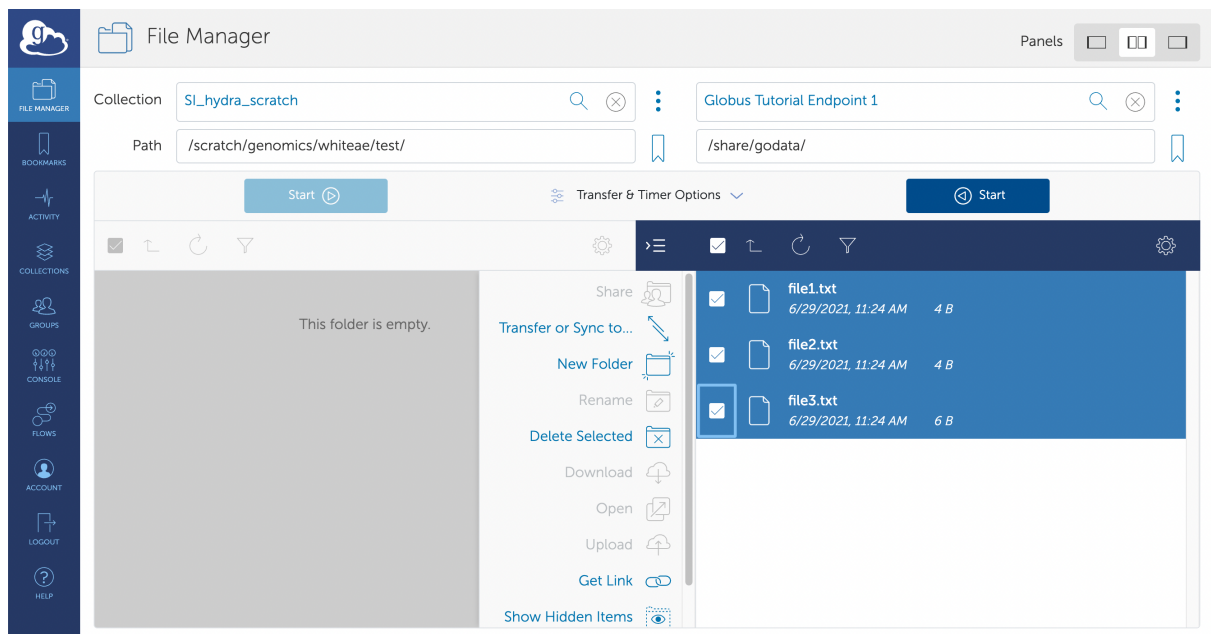
Globus sees files with the same permissions as your Hydra account. What you cannot read on a login node, you cannot read here either.

Transfer

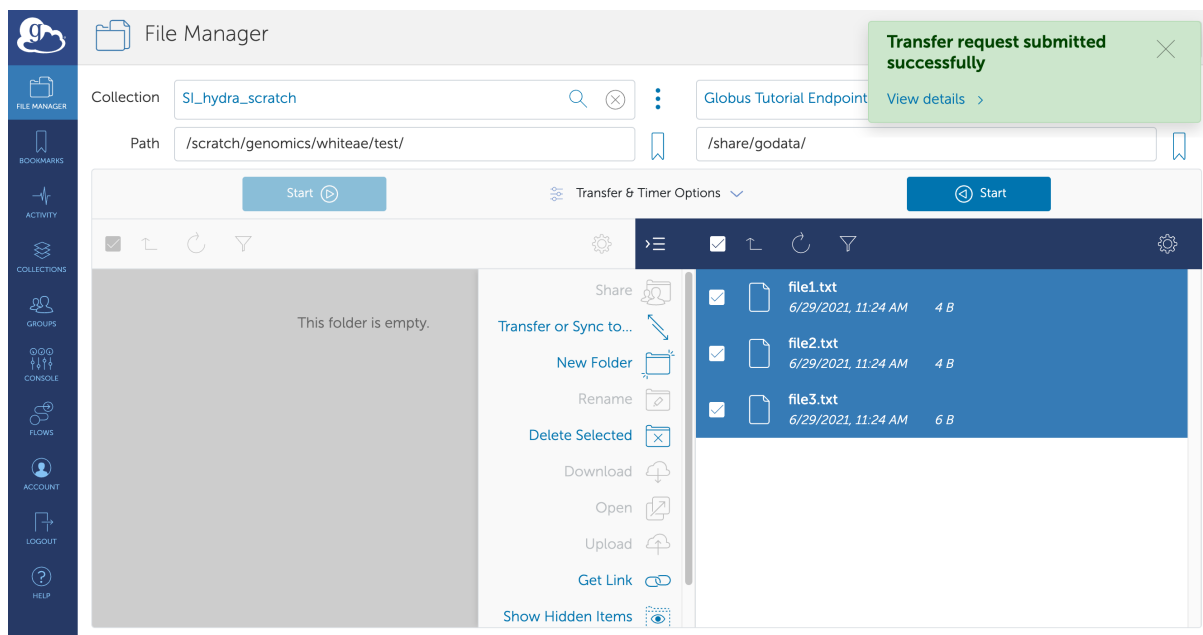
1. With your Hydra directory open in one pane, switch the File Manager to two panes. The layout buttons are at the top right. The middle one gives two panes:



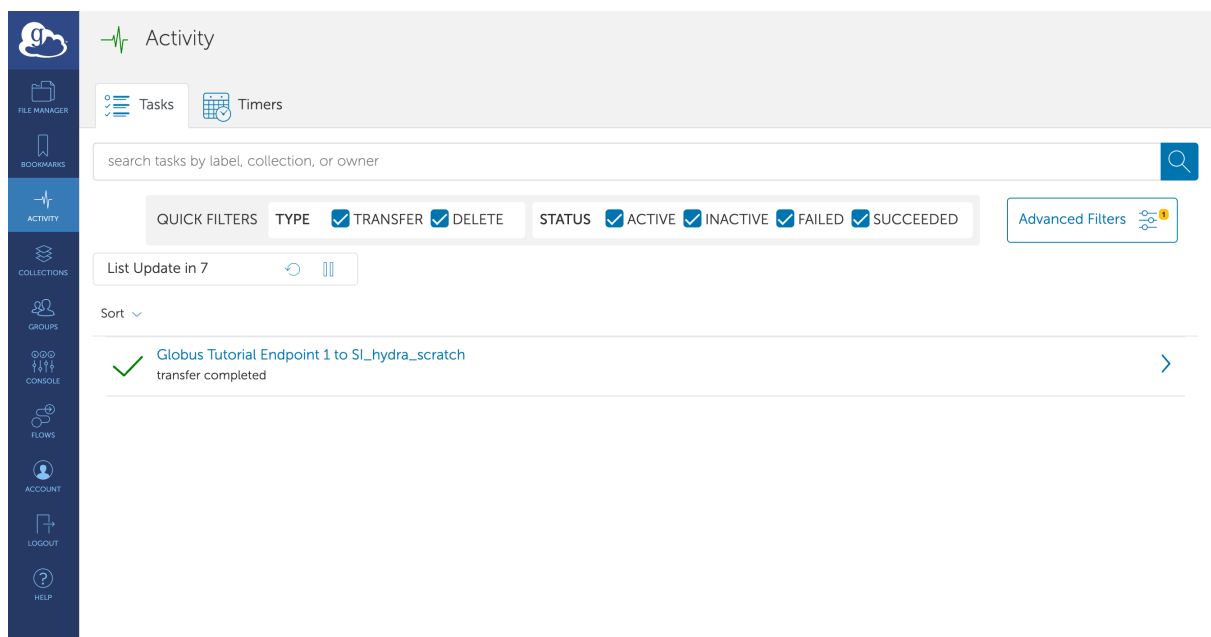
2. In the second pane, open the other collection. That can be another institution's collection, your own computer through Globus Connect Personal, or the **Globus Tutorial Endpoint** collections if you want to try a transfer with nothing at stake.
3. Select files or directories in the source pane and click **Start**. The arrow on the button shows which way the transfer goes:



Globus confirms the request at the top of the page:



- Follow the transfer under **Activity** in the left bar. Globus emails you when it finishes, and also if it fails:



Transfer to and from your own computer

Your computer becomes a collection when it runs Globus Connect Personal. On a machine you administer, install it from Globus's instructions for [macOS](#), [Windows](#) or [Linux](#). On a Smithsonian-administered machine, install it from Software Center. Once it is running, your computer appears in the collection search under the name you gave it, and transfers work as above.

Further reading

- [Smithsonian Globus documentation](#) for sharing, group access and everything beyond Hydra

- [Globus documentation](#) from Globus itself
- [Filesystems](#) for where to put the data once it arrives

Last updated 2026-10-03

rclone and cloud storage

rclone is a command-line program that copies files between a computer and cloud storage such as Dropbox, Google Drive, Amazon S3 and [many other services](#). On Hydra it is the module `tools/rclone`. It is the way to reach a cloud account from Hydra, and the way to get data onto Hydra from a computer that cannot connect to the login nodes, such as one on telework.si.edu.

Setting up a cloud account takes two places. rclone has to be authorized with the cloud service through a web browser, and the login nodes have none, so you run `rclone config` on Hydra, run `rclone authorize` on your own computer, and paste the result back into the Hydra session. The Dropbox walkthrough below shows every prompt.

Set up Dropbox

On Hydra

Load the module and start the configuration. Answer the prompts as shown. `...` marks lines left out.

on Hydra

```
$ module load tools/rclone
$ rclone config
n) New remote
n/s/q> n
name> db
Type of storage to configure.
...
12 / Dropbox
    \ (dropbox)
...
Storage> 12

OAuth Client Id
client_id> [leave blank]
OAuth Client Secret
client_secret> [leave blank]

Edit advanced config? (y/n)
y/n> n

Use web browser to automatically authenticate rclone with remote?
y/n> n
...
config_token>
```

Leave this session at the `config_token>` prompt and go to your own computer.

On your own computer

Download rclone for your operating system from <https://rclone.org/downloads/> and unzip it. In a terminal (Terminal on macOS or Linux, Command Prompt on Windows), change to the unzipped directory and run:

on your computer

```
$ ./rclone authorize "dropbox"
```

On Windows the command is `rclone authorize "dropbox"`, without the `./`.

A browser window opens on Dropbox's sign-in page. Sign in and allow rclone to access the account. The terminal then prints a token of about 150 characters, starting

`{"access_token":`. Copy the whole of it.

Back on Hydra

Paste the token at the waiting prompt and keep the remote:

on Hydra

```
config_token> {"access_token":"...", "token_type":"bearer", "expiry":"..."}
```

```
Configuration complete.
```

```
Keep this "db" remote?
```

```
y/e/d> y
```

```
Current remotes:
```

Name	Type
====	====
db	dropbox

```
e/n/d/r/c/s/q> q
```

rclone saves the remote in `~/.config/rclone/rclone.conf`. If you later want to withdraw rclone's access, open Dropbox's [connected apps](#) settings and remove it.

Other services

Google Drive, Amazon S3 and the rest follow the same two-place pattern. rclone's page for each service shows its prompts, for example [Google Drive](#) and [S3](#). rclone's shared app ID works for most people. If the shared one is throttled, the page for the service says how to register your own.

Copy and sync

List what is in the remote:

```
$ rclone ls db:
```

Copy one file, or a directory, to the cloud:

```
$ rclone copy results.tar db:hydra_backup/  
$ rclone copy /scratch/genomics/USERNAME/project db:project/
```

Copy from the cloud to Hydra:

```
$ rclone copy db:raw/ /scratch/genomics/USERNAME/raw/
```

Make the cloud copy match a directory on Hydra. `-i` asks before each change, which is worth keeping on until you trust the command:

```
$ rclone sync -i /scratch/genomics/USERNAME/project db:project/
```

`--dry-run` on any command shows what it would do and does nothing.

Further reading

- [rclone documentation](#) for every command and option
- [Filesystems](#) for where to put the data on Hydra
- [Warning emails](#) for the login-node limits

Last updated 2026-10-03

Policies & Support

Hydra is a shared system, and we require every user to follow the [usage policies](#). Please read them before you begin. We also provide instructions for [citing Hydra](#) in your publications.

[Cluster status](#) explains where to find the live view of the queues and disks. If you receive a warning email about your jobs, [Warning emails](#) explains what it means and what to do.

Getting help

Email SI-HPC@si.edu. Include the following, so that we can reproduce the problem:

- your Hydra username
- the job ID, from `qstat` for a running job or `qacct+ -j JOBID` for a finished one
- the full command or the job file, and the error message copied exactly
- what you expected to happen instead

Two things go to SI-HPC-Admin@si.edu instead. One is a reply to a warning email about a hosed or oversubscribed job (see [Usage policies](#)). The other is a request to unlock an account whose password expired more than 14 days ago (see [Logging in and passwords](#)). For a new account or VPN access, see [Requesting an account](#).

Before you write

Three questions come up often enough to answer here.

Is a package installed, and how do I see the modules? [Installed modules](#) lists every `bio/` and `tools/` package. `module avail` on a login node lists everything. See [Find and load software](#).

My job finished but I cannot find its output. Without `-cwd` in the job file, the job runs in your home directory and writes its log there. See [Write and submit a job](#).

Can I work interactively? Yes, on a compute node through `qssh`, within the limits of the interactive queue. See [Start an interactive session](#).

Further reading

- [Cluster status](#) to check whether a queue or a disk is the problem before you write
- [Warning emails](#) for what an automated warning means
- [Training](#) for the quarterly introduction workshop

Last updated 2026-10-03

Usage policies

Hydra is an official Smithsonian asset. Everything on it is subject to the Institution's directives on computer and network use, in particular [SD-931](#). You agreed to those directives when you received your Smithsonian network account, and a Hydra account adds the rules on this page. We ask new account holders to reread SD-931, and we expect every Hydra user to keep their computer security awareness training (CSAT) up to date.

Accounts

Anyone with an active Smithsonian network account can request a Hydra account through the [HPC Account Request](#) form. The form collects your name, your unit or department, your supervisor's name and approval, and a sentence or two on the work you plan to do on Hydra. Keep that information current. If your supervisor changes or leaves, or someone else should be contacted about your account and data, tell us.

Accounts for temporary appointments (students, postdocs, fellows, contractors) are renewed once a year. When an account expires we email the user, at the address in their `~/.forward` file, and the supervisor. If neither replies within 30 days, the account's data is subject to deletion.

Two things are prohibited, and either can cost you the account:

- **Sharing credentials.** Your Hydra username and password are yours alone. Nobody else logs in as you, and you log in as nobody else.
- **Borrowing disk space.** Sharing data with collaborators is fine. Storing your data under someone else's account to get around your quota is not. If you need more space or a higher limit for a piece of work, ask us.

We run an introduction to Hydra workshop every quarter. See [Training](#).

Using a shared cluster

As with every system on the Smithsonian network, you have no expectation of privacy in your use of Hydra. Rule 5 of SD-931 on appropriate computer and network use applies.

We reserve the right to suspend, cancel or modify accounts, quotas on the public disks, queue configuration and other settings, without warning, when the integrity of the cluster or its fair use requires it.

Hydra is shared by hundreds of users, and it works only if each of them treats it that way. In practice that means:

- Do not run analyses on the login nodes or the head node. The login nodes are for editing, compiling, transfers and short tests. The [scheduler](#) is how work reaches the compute nodes. Processes that compute on a login node are slowed, then killed.
- Watch your own jobs. You are responsible for monitoring their state and progress, and you should be reachable while they run, so that you can adjust them if something is wrong.

- Start small. Before submitting many similar jobs, run one or a few, check what they used with `qacct`, and size the rest from that.
- Estimate what you need. CPU time, memory and disk space are all limited, and a request far above what a job uses holds the excess back from everyone else. Know roughly how your needs scale with the size of your analysis.
- Everyone's work matters as much as yours. If you wonder whether something is within the spirit of a shared system, ask whether the cluster would still work if dozens of people did the same thing at once.

Systematic abuse of a scarce resource (memory, disk space, and the local SSDs in particular) or bypassing the resource limits is grounds for suspending or cancelling an account.

Disks

Hydra's disks are not archival storage. They are reliable, but the Smithsonian is not responsible for data lost on Hydra, and you should have no expectation that files stored there will survive in the long term. Keep a copy of anything you cannot afford to lose somewhere else.

- `/home` and `/data` have snapshots, and a disaster-recovery copy exists so that the partitions can be rebuilt after a storage failure. That copy is not a backup you can restore files from. See [Backups](#).
- `/scratch/public` is not backed up, and files older than 180 days are removed by the scrubber every week. See [Scrubbed files and restores](#).
- Every public disk has a quota. See [Filesystems](#).
- Put large files and the working data of analyses on `/scratch`.
- When an account expires, its data is subject to deletion 30 days later unless something else has been agreed. A user leaving the Smithsonian either saves their data or hands responsibility for it to their supervisor before they go.

Oversubscribed and inefficient jobs

We monitor the cluster for jobs that use the compute nodes badly, and email the owner. The conditions, and what the email looks like, are under [Warning emails](#).

Condition	Threshold
Oversubscribed job	more than 133% of the CPUs it requested in use
Hosed job	under 10% efficiency after 36 hours
Inefficient job	under 33% of the CPUs it requested in use
Memory over-reserved	reserved more than 2.5 times what it uses
Overloaded node	load above 133% of the node's capacity

For an overloaded node we identify the jobs behind the load, end them, and tell their owners. Most of these are programs that start a thread for every CPU on the node regardless of what was requested. We will help you find the right setting.

What we ask of you when a warning arrives:

- For a hosed or oversubscribed job, reply to SI-HPC-Admin@si.edu within 24 hours where you can, so that we can decide together whether the job should be killed. An oversubscribed job with more than 24 hours still to run should be killed and resubmitted with the right request.
- For an inefficient or memory-over-reserved job, look at what the job is doing, do not ignore the warning, and write to us if you do not understand why it fired or how to fix it for the next job.

When the cluster is busy, or an oversubscription is large, we kill such jobs at our discretion. When the load is above 70%, users with many inefficient jobs have some of them killed automatically, down to 100 unused slots per user, and receive a warning.

Our goal is to get every job through the queue and to help you use a shared system well. An oversubscribed job slows down whoever shares its node. An inefficient or over-reserved job holds CPUs or memory that the scheduler cannot give to the jobs waiting behind it.

Email

We communicate by email, and we expect you to read it. We use it to announce new features, configuration changes and policy changes, and to warn you about jobs that use resources badly, files about to be scrubbed, and accounts about to expire.

All of that goes to the address in the `~/.forward` file in your Hydra home directory. We create the file when the account is created, with your canonical Smithsonian address. If you would rather receive mail elsewhere, put one address on the first line of that file:

```
$ echo you@si.edu > ~/.forward
```

Contact

Questions and requests go to SI-HPC@si.edu. Two things go to SI-HPC-Admin@si.edu. One is a reply to a warning email about a hosed or oversubscribed job. The other is a request to unlock an account whose password expired more than 14 days ago. [Getting help](#) says what to include.

Last updated 2026-10-03

Citing Hydra

Hydra has a DOI, [10.25572/SIHPC](https://doi.org/10.25572/SIHPC). If work you did on Hydra becomes part of a paper, a presentation or a poster, please cite it, either in the references or in the acknowledgments. The citations are how we show what the cluster produces, and that record is what we argue from when we ask for resources.

As a citation:

Smithsonian Institution High Performance Computing Cluster. Smithsonian Institution. <https://doi.org/10.25572/SIHPC>

As an acknowledgment:

The computations in this paper were conducted on the Smithsonian High Performance Cluster (SI/HPC), Smithsonian Institution. <https://doi.org/10.25572/SIHPC>

Use "Some of the computations" if only part of the work ran on Hydra.

We would like to know about the publication. Please [email us](#) the reference. An illustration from the work, with a short caption for a general audience, is welcome too.

Last updated 2026-09-25

Cluster status

The status page at <https://hydra.si.edu/tools/status/> shows what the cluster is doing now and over the past 30 days. It opens only from the SI network or the SI VPN. The page reloads itself every hour, and links at the top switch it to a 5 or 20 minute refresh.

Tab	Shows
Usage	CPU and memory in use on every node; jobs running, queued and available over time; who has jobs running and queued
Warnings	jobs and nodes that need attention
Breakdown by Queue	slots used and free per queue
Avail Slots/Wait Job(s)	where a new job could start, and why waiting jobs wait
Memory Usage	memory reserved and used per node
Resource Limits	the per-user limits and how much of each is in use
Disk Usage & Quota	space and file counts on every filesystem

Two menus under the Usage plots select the snapshot's sort order (name, number of CPUs, usage, load, memory) and the time plot's length (7, 15 or 30 days). You can also highlight one user's jobs. The same choices go in the URL for bookmarking:

```
https://hydra.si.edu/tools/status/?sortby=nCPU&len=15d&user=USERNAME
```

Click a plot to open it at full size. The page also links the list of every installed module, as HTML and as plain text.

The same numbers are available on a login node with `qstat -g c`, `qstat+ -gc` and `qhost`. See [Monitor and manage jobs](#). Planned maintenance and outages are announced in the banner at the top of every page of this site and under [News](#).

Last updated 2026-10-03