<table>
<thead>
<tr>
<th>Quick Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPC Home</td>
</tr>
<tr>
<td>Getting an account</td>
</tr>
<tr>
<td>Getting started on Prince</td>
</tr>
<tr>
<td>Prince How-to Articles</td>
</tr>
<tr>
<td>Logging in Windows</td>
</tr>
<tr>
<td>Mac / Linux</td>
</tr>
<tr>
<td>Clusters and Storage</td>
</tr>
<tr>
<td>Prince (HPC)</td>
</tr>
<tr>
<td>Dumbo (Hadoop)</td>
</tr>
<tr>
<td>Dalma (NYU Abu Dhabi)</td>
</tr>
<tr>
<td>Transferring data to/from the clusters</td>
</tr>
</tbody>
</table>
Users now have a default quota of 500GB on /work, which is NFS mounted on the Mercer login and compute nodes.
As /work is visible on the compute nodes, you can move data between /scratch and /work within your job script.

Moving files between /scratch and /work takes only a single CPU and little memory, however it may take some time. If you find your data movement takes longer than a few minutes, and your job requests more than one CPU or a large amount of memory, we recommend separating the data movement into its own job and using PBS job dependencies to ensure the data movement and the job are executed in the correct order.

You can easily check how long the data movement takes with 'time' (for a single command) or by calling 'date' before and after, for example:

```bash
# ... snippet of script (bash syntax):

# to get data from /work before a run
INPUT_STORE=/work/${USER}/where_my_input_data_is_stored
INPUT_DIR=${SCRATCH}/where_my_model_reads_input_from

# using rsync rather than cp will copy only the files that are not already in
# ${INPUT_DIR}
# since we can copy with one command, use 'time' instead of 'date'
time rsync -av ${INPUT_STORE} ${INPUT_DIR}

# ... run the job in ${SCRATCH} ...

# to copy data back from /scratch to /work after a run:
RESULTS=${SCRATCH}/where_my_job_writes_output
DEST=/work/${USER}/where_i_want_to_keep_it

date

# to take the whole directory:
rsync -av $RESULTS $DEST
# or to take some specific files:
# rsync -av ${RESULTS}/file_I_care_about.* $DEST
# or to take all except for a few files:
# rsync -av -exclude 'junk1 junk2' $RESULTS $DEST

date
```

The example above can be used as a template for using /work with serial jobs, as well as to find how long the data movement takes.

See Managing Data for more information about using rsync to synchronize directories.

If data movement takes longer than a few minutes, we recommend doing the data movement in a separate job. The following workflow illustrates how to achieve this:

1. Prepare the job script in the usual way, running in /scratch

2. Prepare a "fetch input" script which uses rsync to ensure your working area in /scratch has the required data:
3. Prepare a "save results" script which uses rsync to copy the important results back to /work, such as the following:

```
#PBS -S /bin/bash
#PBS -N save_results
## the walltime needed will depend on the amount of data to be transferred
#PBS -l walltime=15:00

RESULTS=${SCRATCH}/where_my_job_writes_output
DEST=/work/${USER}/where_i_want_to_keep_it
# to take the whole directory:
rsync -av $RESULTS $DEST
# or to take some specific files:
# rsync -av $(RESULTS)/file_I_care_about.* $DEST
# or to take all except for a few files:
# rsync -av -exclude 'junk1 junk2' $RESULTS $DEST
# type 'man rsync' for more options
```

4. Submit fetch_input, the actual job, and save_results in a chain, each having a job dependency on the one before:

```
login-0-3$ qsub fetch_input.q
3000193.crunch.local
login-0-3$ qsub -W depend=afterok:3000193 my_job.q
3000194.crunch.local
login-0-3$ qsub -W depend=afterok:3000194 save_results.q
```

Note that the 3000193 and 3000194 above are the job ids returned by each invocation of 'qsub'.

You can do the above in a single line by like:

```
login-0-3$ qsub fetch_input.q | xargs -I {} qsub -W depend=afterok:{} my_job.q | xargs -I {} qsub -W depend=afterok:{} save_results.q
```

The one-line version is three commands joined in a pipeline ({}), using xargs to pass the stdout of one command to the command-line of the next. For more about this type of pipelining see 'man bash' and 'man xargs'. For more about qsub with job dependencies see 'man qsub' or the qsub tutorial.