This guide is intended to help you get started with Swift. You will use swift to generate data, both as named files on disc and as intermediate data and then pass it on to analytical tools. All the programs are deliberately simple because it is intended to show the Swift usage, not the analytical tool usage. Work the examples with the shell scripts then replace them with whatever analytical tool you want to use.

Setup:
Obtain the tutorial package. It is in the svn (directions in the Quickstart (www.ci.uchicago.edu/swift/guides/release-0.94/quickstart/quickstart.html)).

Change directory into the created directory called "tutorial"
This directory contains:

bin: script tools for the tutorial
scripts: Swift scripts for the tutorial

Start:
$ source setup.sh # to add tutorial programs to your PATH ...and then verify: $ random.sh # should be found in your PATH now The tutorial is arranged in parts.

Begin:
$ cd part01
$ cat README
When finished: $ cd ../part02 # etc

In each part, you can type "cleanup" after running a few swift scripts to remove the old logs that build up, and (usually) the output files.

This tutorial takes you through a six step process that demonstrates how to effectively use Swift. It also demonstrates running it on a cluster. Naturally, configuration details will vary more in the demonstration than they did in the six step process.

```
type file;
app (file o) mysim () {
    random stdout=@filename(o);
}
 file f = mysim();
```

The first part is a basic swift program that calls a shell script random.sh . Notice no arguments were passed to the app, but we did use the return value. The return value came from being on the left hand side of the = when the app was called. Notice that although the output file is stored, no filename is given. The file is stored on disc and could be retrieved (it is in the _concurrent folder), but making the file

anonymous (by doing what we did -- not giving it a name) should be done when the file is intended as intermediate data that will not be needed later. This next script shows how to name files. Not only are they easier to find, it is the appropriate style for files that are intended to persist outside the script.

When you look at this file you can see that the output file is specified. In this case it will be named sim.out in folder output. Technically, it doesn't matter if you name files or not since you can find anonymous ones and named ones can be ignored, but searching for files you want and filtering files you don't can impact productivity more than may be initially apparent.

```
type file;
app (file o) mysim () {
    random stdout=@filename(o);
}
  file f<"output/sim.out">; f = mysim();
```

Here we illustrate the use of a loop to run things concurrently. Notice that we do not specify anything about how many instances are running or where they are running. This isn't terribly important as we are running everything locally, but this may be extremely important as we begin to run on more resources.

```
type file;
app (file o) mysim () {
    random stdout=@filename(o);
}
foreach i in [0:9] {
   file f = mysim();
}
```

The exciting part here is that each output file is named. Notice the pattern that allows each file to have a unique name. This pattern can be modified, but some variation mechanism must be provided to prevent file name conflicts.

```
type file;
  app (file o) mysim () {
   random stdout=@filename(o);
}
 foreach i in [0:9] {
   file f <single_file_mapper;
   file=@strcat("output/sim_",i,".out")>;
   f = mysim();
}
```

This illustrates passing the output data on for additional analysis. This technique of specifying the next step can be repeated until the entire workflow is specified.

```
type file;
  app (file o) mysim () {
   random stdout=@filename(o);
}
app (file o) analyze (file s[]) {
   average @filenames(s) stdout=@filename(o);
}
file sims[];
int nsim = @toInt(@arg("nsim","10"));
foreach i in [0:nsim-1] {
   file simout <single_file_mapper;
   file=@strcat("output/sim_",i,".out")>;
   simout = mysim();   sims[i] = simout;
}
file stats<"output/average.out">;
stats = analyze(sims);
```

This demonstrates a more complicated analysis program. Obviously, "complicated" is a relative term here, but the technique is the same whether we're passing one parameter to a simple program or several to a heavyweight analytical tool.

```
type file;
app (file o) mysim2 (int timesteps) {
   random2 timesteps stdout=@filename(o);
}
app (file o) analyze (file s[]) {
   average @filenames(s) stdout=@filename(o);
}
file sims[];
int  nsim = @toInt(@arg("nsim","10"));
int steps = @toInt(@arg("steps","1"));
foreach i in [0:nsim-1] {
   file simout <single_file_mapper;
   file=@strcat("output/sim_",i,".out")>;
   simout = mysim2(steps);   sims[i] = simout;
}
file stats<"output/average.out">;
stats = analyze(sims);
```

This shows the changes necessary to run this script on a cluster. The specific configuration will probably not be particularly useful to you unless you have access to this particular cluster, but it should give you an idea of what information is needed and what format it is expected in.

```
<config>
    <pool handle="localhost">
     <execution provider="local"/>
     <filesystem provider="local"/>
     <workdirectory>
/scratch/midway/{env.USER}/swiftwork</workdirectory>
   </pool>
    <pool handle="westmere">
     <execution provider="coaster" jobmanager="local:slurm"/>
      <!-- Set partition and account here: -->
     <profile namespace="globus" key="queue">
westmere</profile>
     <profile namespace="globus" key="ppn">
12</profile>
     <!-- <profile namespace="globus" key="project">
pi-wilde</profile>
 -->
      <!-- Set number of jobs and nodes per job here: -->
     <profile namespace="globus" key="slots">
1</profile>
     <profile namespace="globus" key="maxnodes">
1</profile>
     <profile namespace="globus" key="nodegranularity">
1</profile>
     <profile namespace="globus" key="jobsPerNode">
12</profile>
 <!-- apps per node! -->
     <profile namespace="karajan" key="jobThrottle">
.11</profile>
 <!-- eg .11 ->
 12 -->
      <!-- Set estimated app time (maxwalltime) and requested job time (maxtime) here:
-->
     <profile namespace="globus" key="maxWalltime">
00:15:00</profile>
     <profile namespace="globus" key="maxtime">
1800</profile>
  <!-- in seconds! -->
      <!-- Set data staging model and work dir here: -->
     <filesystem provider="local"/>
     <workdirectory>
/scratch/midway/{env.USER}/swiftwork</workdirectory>
      <!-- Typically leave these constant: -->
     <profile namespace="globus" key="slurm.exclusive">
false</profile>
     <profile namespace="globus" key="highOverAllocation">
100</profile>
     <profile namespace="globus" key="lowOverAllocation">
100</profile>
     <profile namespace="karajan" key="initialScore">
10000</profile>
   </pool>
  </config>
```

```
type file;
app (file o) mysim4 (int timesteps, int range, int count) {
   random4 timesteps range count stdout=@filename(o);
}
app (file o) analyze (file s[]) {
   average @filenames(s) stdout=@filename(o);
}
  file sims[];
int  nsim  = @toInt(@arg("nsim",  "10"));
int  steps = @toInt(@arg("steps", "1"));
int  range = @toInt(@arg("range", "100"));
int  count = @toInt(@arg("count", "10"));
foreach i in [0:nsim-1] {
   file simout <single_file_mapper;
   file=@strcat("output/sim_",i,".out")>;
   simout = mysim4(steps,range,count);
   sims[i] = simout;
}
file stats<"output/average.out">; stats = analyze(sims);
```

```
westmere  random4  random4.sh
localhost average  avg.sh
```

Notice that none of the changes in the .swift file actually have anything to do with running on a cluster. The additional parameters are there to allow us to better see performance changes, not running on a server. The changes that allow us to run on the cluster are in the tc and sites files.

We've now created a simple workflow with Swift. You can add programs to the workflow and manage their output. TO make them more complicate you need only call more advanced utilities and repeat the techniques you have learned. There are a lot more things you can learn to make Swift work better, and you will find those in the user manual. Go ahead and experiment with it a little.