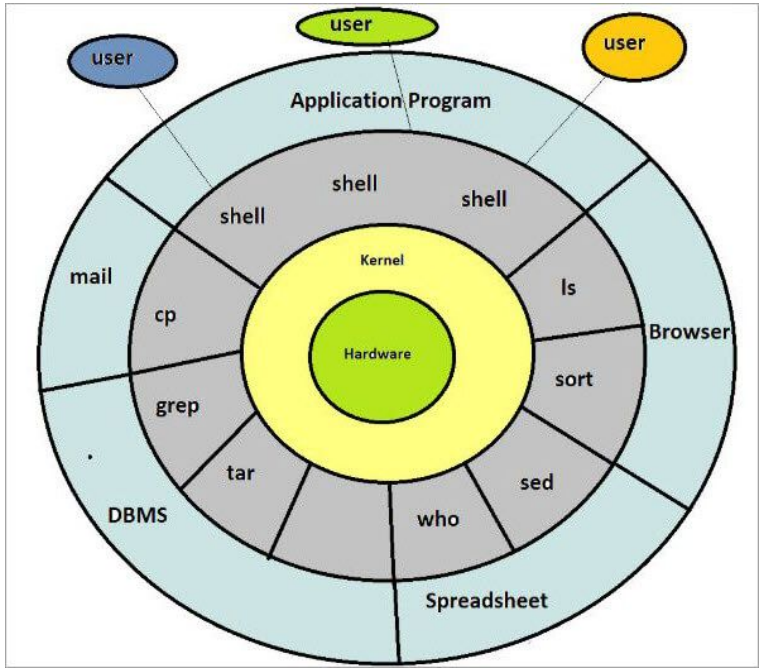
Unix tips and tricks

Cluster management, scripting and more!

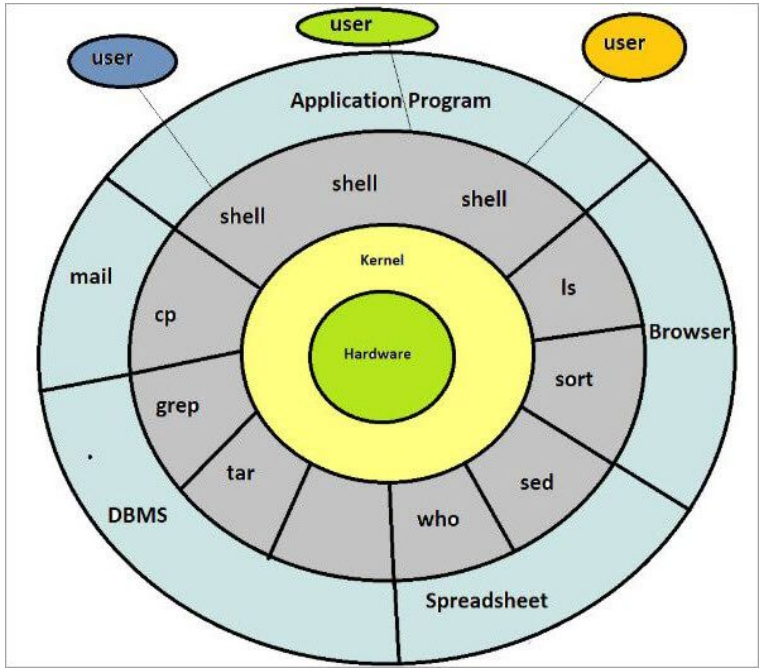
What is “Unix”?

- A family of operating systems
- The basis for many popular systems:
  - Mac OS X
  - CfN cluster
  - Neuroimaging programs like FSL, XCP



What is “Unix”?

- Kernel: core of the OS that interacts directly with hardware
- Shell: program that takes input from keyboard (command line) and passes it to kernel
- bash: Bourne Again SHell, enhanced version of the original shell program, written by Steve Bourne



Overview

Basic commands

Cluster management

Intro to scripting

Basic commands

Basic Unix commands (vs. DOS)

• **cat** *file*	**Concatenate or type out a file**
cat *file1 file2* **...**	**Type out a number of files**
• **cd** *directory1*	**Change current directory to** *directory1*
cd /usr/bin	**Change current directory to /usr/bin**
cd	**Change back to your home directory**
• **clear**	**Clear the current screen**
• **cp** *file1 file2*	**Copy** *file1* **to** *file2*
cp *file1 file2* **...** *dir*	**Copy a number of files to a directory**
• **ls**	**List the files in the current directory**
ls /usr/bin	**List the files in the /usr/bin directory**
• **lpr** *file1*	**Print** *file1* **out**
lpr *file1 file2* **...**	**Print a number of files out**
• **more** *file*	**Look at the content of a file with paging, use 'q' to get out**
• **mkdir** *directory*	**Create a directory**
• **mv** *file1 file2*	**Move** *file1* **to** *file2*, **like rename.**
mv *file1 file2* **...** *dir*	**Move a number of files into a directory**
mv *dir1 dir2*	**Move or rename a directory**

Unix/Linux Command Reference

FOSSwire.com

File Commands

- **ls** – directory listing
- **ls -al** – formatted listing with hidden files
- **cd *dir*** - change directory to *dir*
- **cd** – change to home
- **pwd** – show current directory
- **mkdir *dir*** – create a directory *dir*
- **rm *file*** – delete *file*
- **rm -r *dir*** – delete directory *dir*
- **rm -f *file*** – force remove *file*
- **rm -rf *dir*** – force remove directory *dir* *
- **cp *file1 file2*** – copy *file1* to *file2*
- **cp -r *dir1 dir2*** – copy *dir1* to *dir2*; create *dir2* if it doesn't exist
- **mv *file1 file2*** – rename or move *file1* to *file2* if *file2* is an existing directory, moves *file1* into directory *file2*
- **ln -s *file link*** – create symbolic link *link* to *file*
- **touch *file*** – create or update *file*
- **cat > *file*** – places standard input into *file*
- **more *file*** – output the contents of *file*
- **head *file*** – output the first 10 lines of *file*
- **tail *file*** – output the last 10 lines of *file*
- **tail -f *file*** – output the contents of *file* as it grows, starting with the last 10 lines

Process Management

- **ps** – display your currently active processes
- **top** – display all running processes
- **kill *pid*** – kill process id *pid*
- **killall *proc*** – kill all processes named *proc* *
- **bg** – lists stopped or background jobs; resume a stopped job in the background
- **fg** – brings the most recent job to foreground
- **fg *n*** – brings job *n* to the foreground

File Permissions

chmod *octal file* – change the permissions of *file* to *octal*, which can be found separately for user, group, and world by adding:

- 4 – read (r)
- 2 – write (w)
- 1 – execute (x)

Examples:
chmod 777 – read, write, execute for all
chmod 755 – rwx for owner, rx for group and world
For more options, see **man chmod**.

SSH

- **ssh *user@host*** – connect to *host* as *user*
- **ssh -p *port user@host*** – connect to *host* on port *port* as *user*
- **ssh-copy-id *user@host*** – add your key to *host* for *user* to enable a keyed or passwordless login

Searching

- **grep *pattern files*** – search for *pattern* in *files*
- **grep -r *pattern dir*** – search recursively for *pattern* in *dir*
- ***command* | grep *pattern*** – search for *pattern* in the output of *command*
- **locate *file*** – find all instances of *file*

System Info

- **date** – show the current date and time
- **cal** – show this month's calendar
- **uptime** – show current uptime
- **w** – display who is online
- **whoami** – who you are logged in as
- **finger *user*** – display information about *user*
- **uname -a** – show kernel information
- **cat /proc/cpuinfo** – cpu information
- **cat /proc/meminfo** – memory information
- **man *command*** – show the manual for *command*
- **df** – show disk usage
- **du** – show directory space usage
- **free** – show memory and swap usage
- **whereis *app*** – show possible locations of *app*
- **which *app*** – show which *app* will be run by default

Compression

- **tar cf *file.tar files*** – create a tar named *file.tar* containing *files*
- **tar xf *file.tar*** – extract the files from *file.tar*
- **tar czf *file.tar.gz files*** – create a tar with Gzip compression
- **tar xzf *file.tar.gz*** – extract a tar using Gzip
- **tar cjf *file.tar.bz2*** – create a tar with Bzip2 compression
- **tar xjf *file.tar.bz2*** – extract a tar using Bzip2
- **gzip *file*** – compresses *file* and renames it to *file.gz*
- **gzip -d *file.gz*** – decompresses *file.gz* back to *file*

Network

- **ping *host*** – ping *host* and output results
- **whois *domain*** – get whois information for *domain*
- **dig *domain*** – get DNS information for *domain*
- **dig -x *host*** – reverse lookup *host*
- **wget *file*** – download *file*
- **wget -c *file*** – continue a stopped download

Installation

Install from source:
./configure
make
make install
dpkg -i *pkg.deb* – install a package (Debian)
rpm -Uvh *pkg.rpm* – install a package (RPM)

Shortcuts

- **Ctrl+C** – halts the current command
- **Ctrl+Z** – stops the current command, resume with **fg** in the foreground or **bg** in the background
- **Ctrl+D** – log out of current session, similar to **exit**
- **Ctrl+W** – erases one word in the current line
- **Ctrl+U** – erases the whole line
- **Ctrl+R** – type to bring up a recent command
- **!!** - repeats the last command
- **exit** – log out of current session

* use with extreme caution.

Basic commands - Navigation

Ctrl + a : move to beginning of line

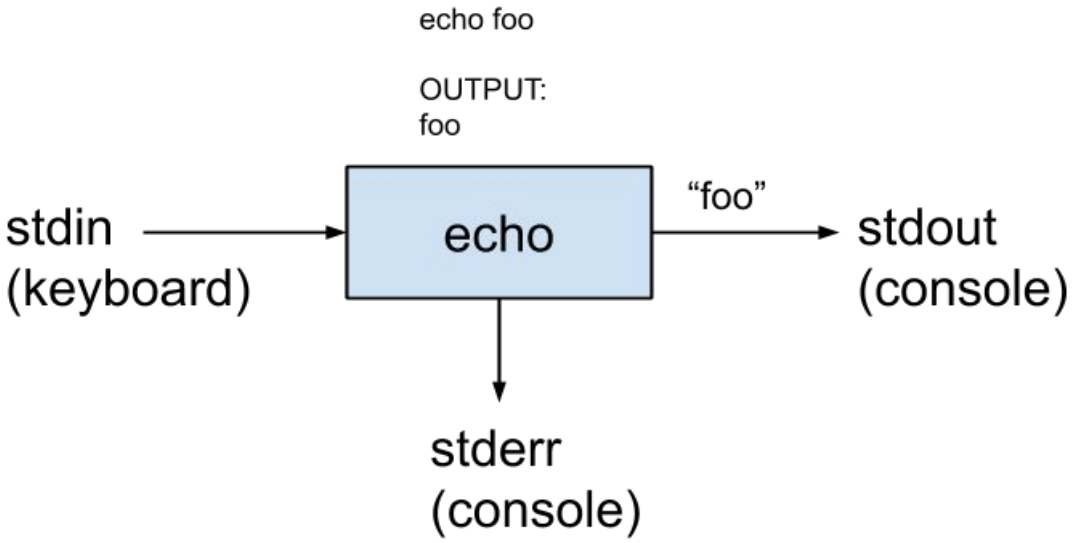
Ctrl + e : move to end of line

Highlight : copy (Mac -> command + c)

```
[dseok@chead ~]$
```

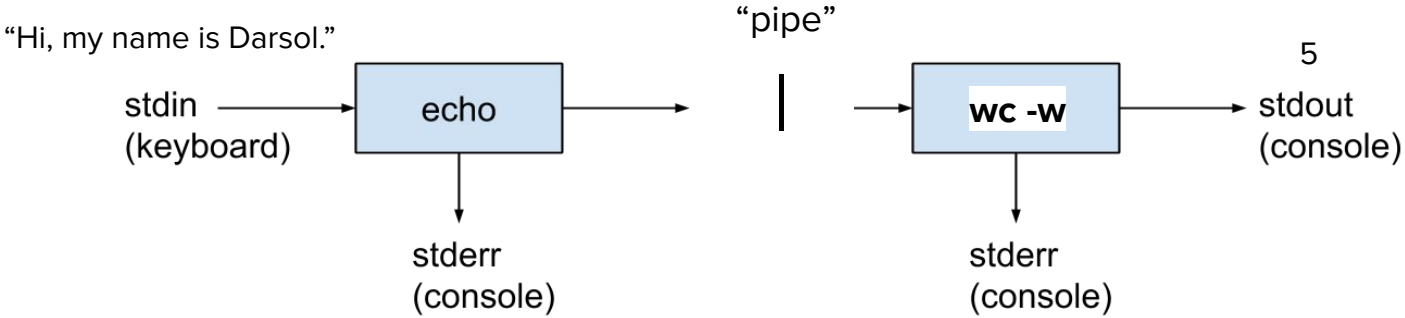
Middle mouse click : paste (Mac -> command + v)

Basic commands - stdin, stdout, stderr



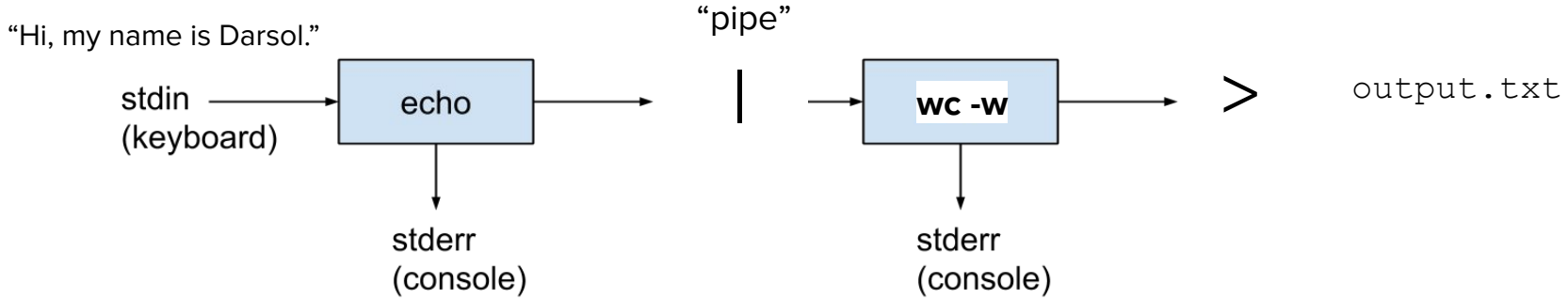
Basic commands - pipes and redirection

```
[dseok@chead ~]$ echo "Hi, my name is Darsol." | wc -w
5
```



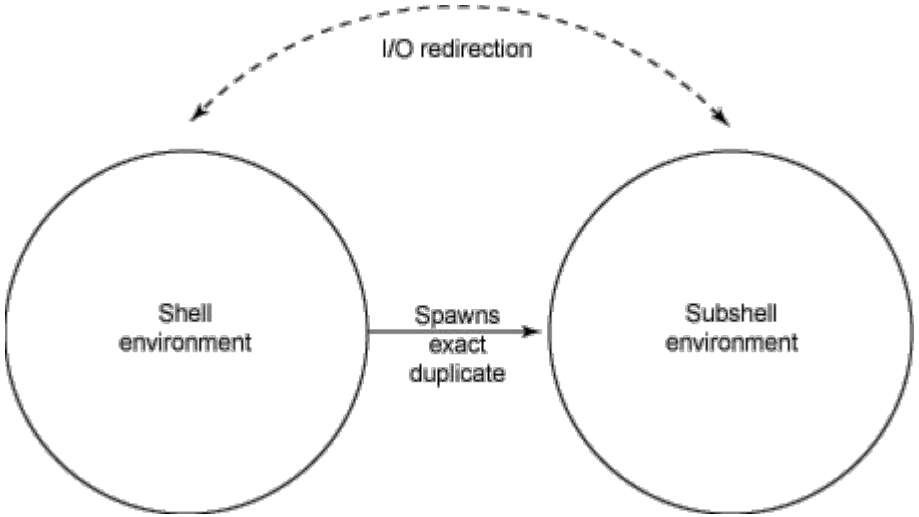
Basic commands - pipes and redirection

```
[dseok@chead ~]$ echo "Hi, my name is Darsol" | wc -w > output.txt
[dseok@chead ~]$ cat output.txt
5
```



Basic commands - pipes and redirection - subshells

```
var=$(ls | wc -w)
```



Basic commands - pipes and redirection

```
# check that the subject has an acceptable status
status_col=$(head -1 ${sublist} | tr ',' '\n' | grep -n '"status"' | awk -F ':' '{print $1}')
status=$(grep ${basesub} ${sublist} | awk -F ',' -v var=${status_col} '{print $var}' | sed 's/"//g')
```

Basic commands - the PATH

What happens when you type commands like `ls`, `cd` and `mkdir`?

The shell searches through all the directories in your PATH

PATH is defined in a file called your bash_profile

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

## CFNAPPS - defined in /etc/profile.d/cfn-common.sh

## R & RStudio
# Check that this is same version pointed to by the ANTsR build path
# NOTE: Define R_LIBS to include packages that aren't in R/library
#PATH=$PATH:$CFNAPPS/R/R-3.1.1/bin
PATH=$PATH:$CFNAPPS/R/R-3.2.3/bin

## RStudio
# NOTE: Define R_LIBS to include packages that aren't in R/library
PATH=$PATH:$CFNAPPS/rstudio/rstudio-0.98.1091/bin/

## Pandoc
# Using the install that comes with rstudio because of
#  difficulty building on its own.
# I believe this will use environment settings (i.e. PATH and R_LIBS)
#  to run R as needed.
# Put it ahead of PATH so old default install isn't used.
PATH=$CFNAPPS/pandoc/1.12.4.2-in-rstudio/:$PATH
```

Basic commands - using aliases to set up shortcuts

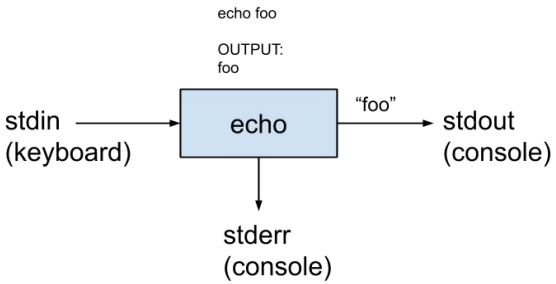
You can set up aliases (shortcuts) in your bash_profile

```
alias hcp='cd /data/jag/cnds/connectome_U01/hcpPipeline'
alias beh='cd /data/jag/cnds/connectome_U01/MRIbehavior/task_timing_fil
es'
alias ccbt='cd /data/jag/cnds/ccbt'
alias stress='cd /data/jag/cnds/stressInflammation'
alias jux='cd /data/jux/cnds'
alias cndsapps='cd /data/jag/cnds/applications'

alias queueup='qalter -u dseok -q all.q,basic.q,himem.q'

alias transform_results='/data/jag/cnds/stressInflammation/scripts/rest
ing/sca/third_lev/transform_third_lev_results.sh $(pwd)'
alias transform_randomise='/data/jag/cnds/stressInflammation/scripts/re
sting/sca/randomise/transform_randomise_results.sh $(pwd)'
alias prepare_randomise='/data/jag/cnds/ccbt/scripts/task/third_lev/ran
domise/prepare_randomise_results.sh $(pwd)'
alias transform_ica='/data/jag/cnds/stressInflammation/scripts/resting/
ica/transform_ica.sh $(pwd)'
```

Basic commands - background processes, nohup




```
[dseok@chead ~]$ jobs -l
[1]   6946 Running                 sleep 20 &
[2]   6947 Running                 sleep 20 &
[3]   6948 Running                 sleep 20 &
[4]   6949 Running                 sleep 20 &
[5]   6950 Running                 sleep 20 &
[6]   6951 Running                 sleep 20 &
[7]   6952 Running                 sleep 20 &
[8]   6953 Running                 sleep 20 &
[9]   6954 Running                 sleep 20 &
[10]   6955 Running                 sleep 20 &
[11]   6956 Running                 sleep 20 &
[12]   6957 Running                 sleep 20 &
[13]   6958 Running                 sleep 20 &
[14]   6959 Running                 sleep 20 &
[15]   6960 Running                 sleep 20 &
[16]   6961 Running                 sleep 20 &
[17]   6962 Running                 sleep 20 &
[18]   6963 Running                 sleep 20 &
[19]-  6964 Running                 sleep 20 &
[20]+  6965 Running                 sleep 20 &
```

- Typically, while a process is running, the command line is not returned
- Using `&` after any command will place the process in the background
- `jobs -l` will show list of running jobs
- `kill [process_id]` will stop a running process
- `nohup [command]` will run the command, will not stop if you log off

Basic commands - background processes, nohup

```
for seed in $(cat seeds.txt); do
./wrapper_run_analysis.sh ${seed} > ${seed}.log & done
```

```
[dseok@chead randomise]$ for seed in $(cat seeds.txt); do ./wrapper_run
_randomise.sh group_diff ${seed} baseline > ${seed}.log & done
```

Basic commands - background processes, nohup

```
nohup [command] > [log_file] &
```

Running XCP:

```
nohup ${XCPEDIR}/xcpEngine -d design.dsn -c cohort.csv -o
output_dir > output_dir.log &
```

Basic commands - useful tools

`awk` – when your file or string has delimiters

`sed` – search and replace, print specific lines

`grep` – search and output

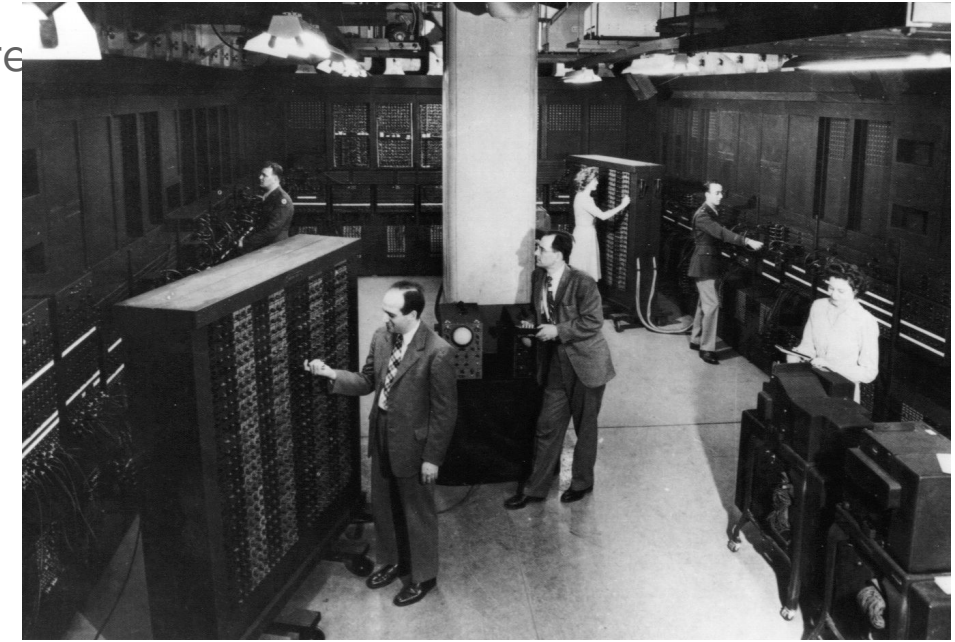
`bc` – integer and floating point calculations

`find` – finding files

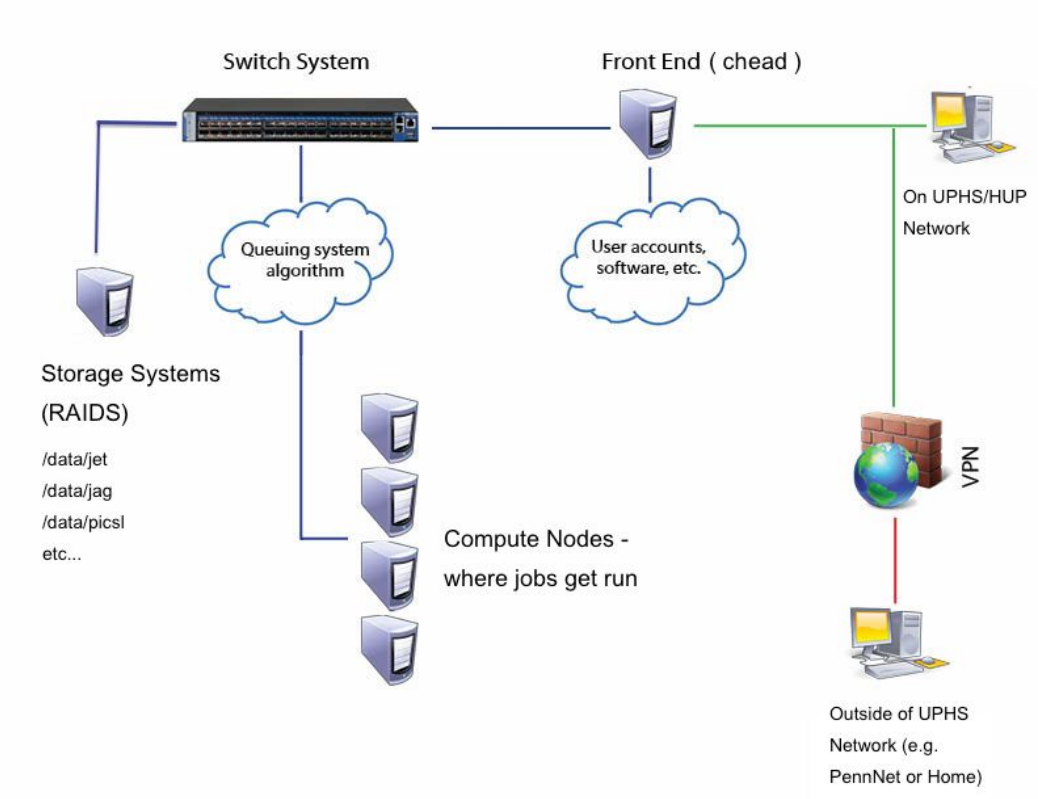
`... and many more`

Cluster management

Good management of your compute core can make you more efficient!



Cluster management - overview

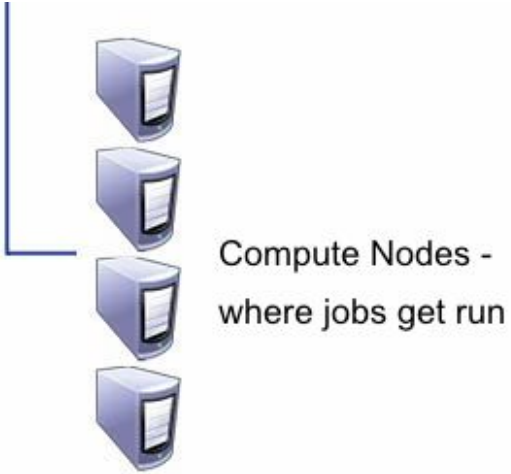


Cluster management - interactive and batch nodes

Two flavors of nodes:

1. Interactive: directly login to a compute node. Use `qlogin`

2. Batch: submit a job to a compute node, stay on head node. Use `qsub`



Cluster management - interactive and batch nodes

Which node to use?

Cluster management - interactive and batch nodes

Which node to use?

Head node:

- Processes using > 5GB or lasting > 5 mins will be killed
- File management, submitting jobs, editing scripts, viewing images

Cluster management - interactive and batch nodes

Which node to use?

Head node:

- Processes using > 5GB or lasting > 5 mins will be killed
- File management, submitting jobs, editing scripts, viewing images

Interactive node:

- Directly log into compute node
- 8 slots, 230GB allocated
- Conducting analyses with GUIs - MATLAB
- Live debugging of scripts

Cluster management - interactive and batch nodes

Which node to use?

Head node:

- Processes using > 5GB or lasting > 5 mins will be killed
- File management, submitting jobs, editing scripts, viewing images

Interactive node:

- Directly log into compute node
- 8 slots, 230GB allocated
- Conducting analyses with GUIs - MATLAB
- Live debugging of scripts

Compute node:

- Submit a script (“job”) to the cluster
- 70 slots, ~600GB allocated
- Any other job

Cluster management - queues

In addition to interactive and batch, nodes come in different flavors:

`all.q` - default queue, standard cpu speed, 30GB memory limit. (220 GB, 32 slots)

`himem.q` - queue with higher memory limit - up to 220GB. (220GB, 6 slots)

`basic.q` - older set of compute cores, 19GB memory limit, slower. (130GB, 32 slots)

`gpu.q` - for use with GPU applications

Cluster management - queues

In addition to interactive and batch, nodes come in different flavors:

`all.q` - default queue, standard cpu speed, 30GB memory limit. (220 GB, 32 slots)

`himem.q` - queue with higher memory limit - up to 220GB. (220GB, 6 slots)

`basic.q` - older set of compute cores, 19GB memory limit, slower. (130GB, 32 slots)

`gpu.q` - for use with GPU applications

Also, **`qlogin.q`** (30GB, 5 slots) and **`qlogin.himem.q`** (220GB, 3 slots)

Cluster management - queues

By default, your account will only use **`all.q`** and **`qlogin.q`**

If your job isn’t being scheduled, consider using other queues!

If you need to run many jobs as quickly possible, use all of the queues!

Cluster management - commands

`qsub test_script.sh` : submit a job

- `-l h_vmem=10G,s_vmem=9.5g` : set memory limits
- `-q all.q,basic.q,himem.q` : set queues
- `-o /data/jag/cnds/logs/test_script_\$JOB_ID.o` : set stdout
- `-e /data/jag/cnds/logs/test_script_\$JOB_ID.e` : set stderr
- `-j y` : merge stdout and stderr
- `-cwd` : launch in current working directory
- `-m e -M dseok@sas.upenn.edu` : send email when job is completed

Cluster management - commands

`qsub test_script.sh` : submit a job

- `-l h_vmem=10G,s_vmem=9.5g` : set memory limits
- `-q all.q,basic.q,himem.q` : set queues
- `-o /data/jag/cnds/logs/test_script_\$JOB_ID.o` : set stdout
- `-e /data/jag/cnds/logs/test_script_\$JOB_ID.e` : set stderr
- `-j y` : merge stdout and stderr
- `-cwd` : launch in current working directory
- `-m e -M dseok@sas.upenn.edu` : send email when job is completed

```
qlogin -q qlogin.himem.q -l h_vmem=40G,s_vmem=40G
```

Cluster management - commands

`qstat` : show all your running jobs

`cfn-resources` : show your available resources

`qalter -q all.q,basic.q,himem.q [job-id]` : change queue of job-id

`qdel [job-id]` : kill a running or queued job

Cluster management - example

Objective: change the queue of jobs started on 06/12/2019 that are currently queued (state = qw)

```
6724320 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 basic.q@compute-2-7.local          1
6724321 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 basic.q@compute-2-5.local          1
6724322 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 basic.q@compute-2-11.local         1
6724323 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 basic.q@compute-1-3.local          1
6724324 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 basic.q@compute-1-11.local         1
6724325 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 basic.q@compute-2-9.local          1
6724326 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 basic.q@compute-2-15.local         1
6724327 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 basic.q@compute-2-6.local          1
6724328 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 basic.q@compute-1-5.local          1
6724329 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 basic.q@compute-2-0.local          1
6724330 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 himem.q@compute-0-20.local         1
6724331 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 himem.q@compute-0-19.local         1
6724332 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 himem.q@compute-0-19.local         1
6724333 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 himem.q@compute-0-20.local         1
6724334 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 himem.q@compute-0-20.local         1
6724335 0.50500 sleep.sh   dseok        r     06/12/2019 09:56:16 himem.q@compute-0-19.local         1
6724336 0.50500 sleep.sh   dseok        qw    06/12/2019 09:56:12                                    1
6724337 0.50500 sleep.sh   dseok        qw    06/12/2019 09:56:12                                    1
6724338 0.50500 sleep.sh   dseok        qw    06/12/2019 09:56:12                                    1
6724339 0.50500 sleep.sh   dseok        qw    06/12/2019 09:56:12                                    1
6724340 0.50500 sleep.sh   dseok        qw    06/12/2019 09:56:12                                    1
6724341 0.50500 sleep.sh   dseok        qw    06/12/2019 09:56:12                                    1
6724342 0.50500 sleep.sh   dseok        qw    06/12/2019 09:56:12                                    1
```

Cluster management - example

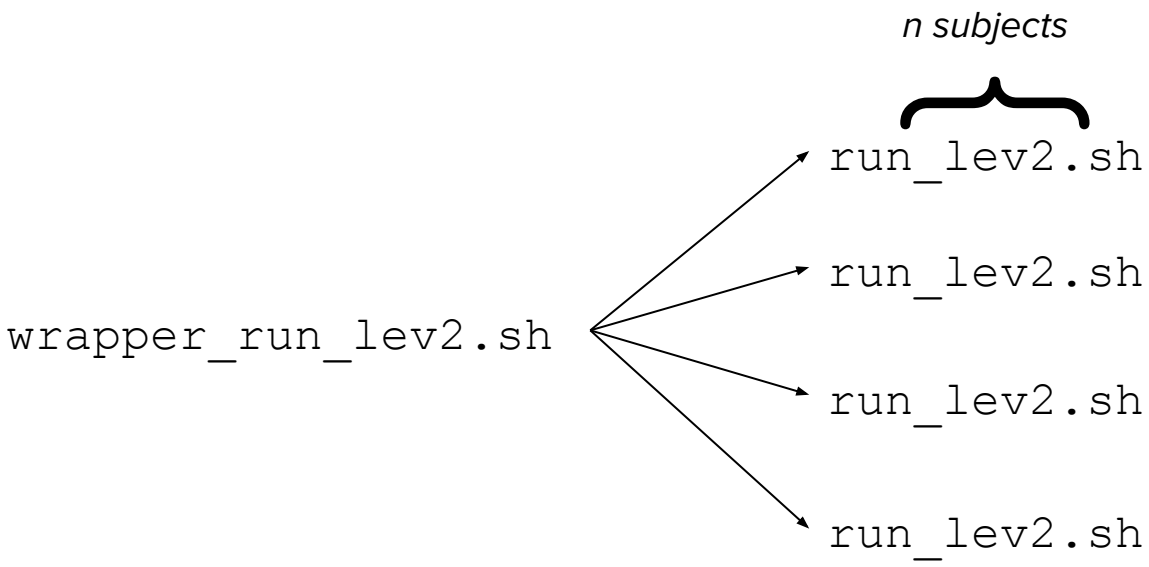
```
[dseok@chead ~]$ qalter $(qstat | grep 06/12/2019 | grep qw | awk '{print $1}') -q basic.q
```

Intro to scripting

Script - a series of commands that performed sequentially to perform a task

Intro to scripting - example script

Script to run “level 2”, within subjects analysis



Intro to scripting - shebang

```
#!/bin/bash

# wrapper for run_lev2.sh. Will submit run_lev2.sh to the basic queue.

# args:
XCP_OUTPUT_NAME=${1}
LEV2_OUTPUT_NAME=${2}
```

`#!` : the shebang, a character sequence that indicates which interpreter to use

- `#!/usr/bin/env Rscript` : use `env` to launch interpreter for Rscript
- `#!/usr/bin/python` : use this version of python to interpret this script

Intro to scripting - comments

```
#!/bin/bash

# wrapper for run_lev2.sh. Will submit run_lev2.sh to the basic queue.

# args:
XCP_OUTPUT_NAME=${1}
LEV2_OUTPUT_NAME=${2}
```

You can use # to insert comments - human readable text that helps people (including yourself!) to read your code

Intro to scripting - arguments

```
#!/bin/bash

# wrapper for run_lev2.sh. Will submit run_lev2.sh to the basic queue.

# args:
XCP_OUTPUT_NAME=${1}
LEV2_OUTPUT_NAME=${2}
```

When a script is launched as an executable, positional arguments are assigned to numbers

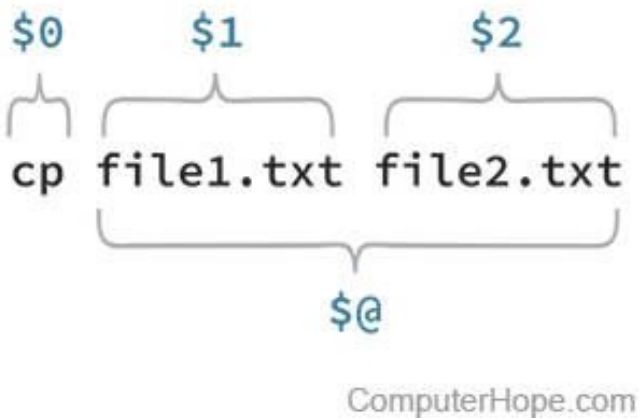
Intro to scripting - arguments

```
#!/bin/bash

# wrapper for run_lev2.sh. Will submit run_lev2.sh to the basic queue.

# args:
XCP_OUTPUT_NAME=${1}
LEV2_OUTPUT_NAME=${2}
```

When a script is launched as an executable, positional arguments are assigned to numbers



Intro to scripting - variables

```
PROJECT=/data/jag/cnds/ccbt
XCPDIR=/data/jux/cnds/xcp/outputs/${XCP_OUTPUT_NAME}
OUTPUTDIR=/data/jux/cnds/fsl/second_lev/${LEV2_OUTPUT_NAME}

mkdir -p ${OUTPUTDIR}
```

I like to assign variables to everything, so script is more portable

- Translating script to another project/data structure
- When your data structures change
- Also makes your script more readable

Intro to scripting - for loop

```
for sub in ${XCPDIR}/NDAR*; do
        for tp in $(ls ${sub}); do
                # check that sub and tp not already present
                basesub=$(basename ${sub})
                if [ -d ${OUTPUTDIR}/${basesub}_${tp}.gfeat ]; then
                        echo ${basesub}_${tp} has already completed second level
analysis.
                        continue
                fi
```

The `for` loop is a control structure to iterate over a set of things

Intro to scripting - for loop

```
for sub in ${XCPDIR}/NDAR*; do
        for tp in $(ls ${sub}); do
                # check that sub and tp not already present
                basesub=$(basename ${sub})
                if [ -d ${OUTPUTDIR}/${basesub}_${tp}.gfeat ]; then
                        echo ${basesub}_${tp} has already completed second level
analysis.
                        continue
                fi
```

```
for [thing] in [list_of_things]; do

    [stuff]

done
```

Intro to scripting - if statement

```
for sub in ${XCPDIR}/NDAR*; do
        for tp in $(ls ${sub}); do
                # check that sub and tp not already present
                basesub=$(basename ${sub})
                if [ -d ${OUTPUTDIR}/${basesub}_${tp}.gfeat ]; then
                        echo ${basesub}_${tp} has already completed second level
analysis.
                        continue
                fi
```

`if` statements can be used to check if a condition is met, and then perform some action

`continue` is used to skip to the next item in your `for` loop

Intro to scripting - if statement

```
for sub in ${XCPDIR}/NDAR*; do
        for tp in $(ls ${sub}); do
                # check that sub and tp not already present
                basesub=$(basename ${sub})
                if [ -d ${OUTPUTDIR}/${basesub}_${tp}.gfeat ]; then
                        echo ${basesub}_${tp} has already completed second level
analysis.
                        continue
                fi
```

```
if [ some_condition_to_meet ]; then

    do_something

fi
```

Intro to scripting - multiple conditions

```
                # check that xcp has completed
                if [ ! -d $(echo ${sub}/${tp}/*/task/contrasts | awk '{print $1}') ] \
                || [ $(ls ${sub}/${tp} | wc -w) -ne $(echo ${sub}/${tp}/*/task/contrasts | wc -w) ]; then
                        echo "ERROR: ${basesub}_${tp} XCP run has failed or not completed."
                        continue
                fi

                echo "Submitting ${sub} - ${tp}"
                # otherwise, will we submit to run_lev2.sh
                qsub -j y -o ${PROJECT}/scripts/task/logs/run_lev2/${basesub}_${tp}_\$JOB_ID.stdout -cwd ${P
ROJECT}/scripts/task/second_lev/run_lev2.sh ${OUTPUTDIR} ${sub} ${tp}
        done
done
```

You can string multiple conditions with logical connectors. Most common:

`&&` : and, both must be true

`||` : or, either one or the other must be true

Intro to scripting - submitting “sub-script”

```
                # check that xcp has completed
                if [ ! -d $(echo ${sub}/${tp}/*/task/contrasts | awk '{print $1}') ] \
                || [ $(ls ${sub}/${tp} | wc -w) -ne $(echo ${sub}/${tp}/*/task/contrasts | wc -w) ]; then
                        echo "ERROR: ${basesub}_${tp} XCP run has failed or not completed."
                        continue
                fi

                echo "Submitting ${sub} - ${tp}"
                # otherwise, will we submit to run_lev2.sh
                qsub -j y -o ${PROJECT}/scripts/task/logs/run_lev2/${basesub}_${tp}_\$JOB_ID.stdout -cwd ${P
ROJECT}/scripts/task/second_lev/run_lev2.sh ${OUTPUTDIR} ${sub} ${tp}
        done
done
```

Finally, we run our “sub-script”, `run_lev2.sh`

We pass our subject-specific parameters to the next script

Intro to scripting - submitting “sub-script”

```
#!/bin/bash

# runs second level analysis

OUTPUTDIR=${1}
sub=${2}
tp=${3}


LEV2_OUTPUT_NAME=$(basename ${OUTPUTDIR})
basesub=$(basename ${sub})
fsf_file=TEMP${LEV2_OUTPUT_NAME}_${basesub}_${tp}.fsf

# edit in the file names of the feat directories
feats=$(ls ${sub}/${tp}/*/task/fsl | grep "feat")
```

Receiving parameters from wrapper script

Thank you!