Introduction to Linux / Using the Ada Cluster

T. Mark Huang

TEXAS A&M UNIVERSITY
HIGH PERFORMANCE RESEARCH COMPUTING

For HPRC Bioinformatics Workshop
Summer 2016 @ CVM

Outline

- Usage Policies
- (Brief) Introduction to Linux
- Introduction to Using the Ada Cluster
  - Hardware Overview of Ada
  - Accessing Ada
  - Filesystems and User Directories
  - Computing Environment
  - Batch Processing
  - Common Problems
- Need Help?

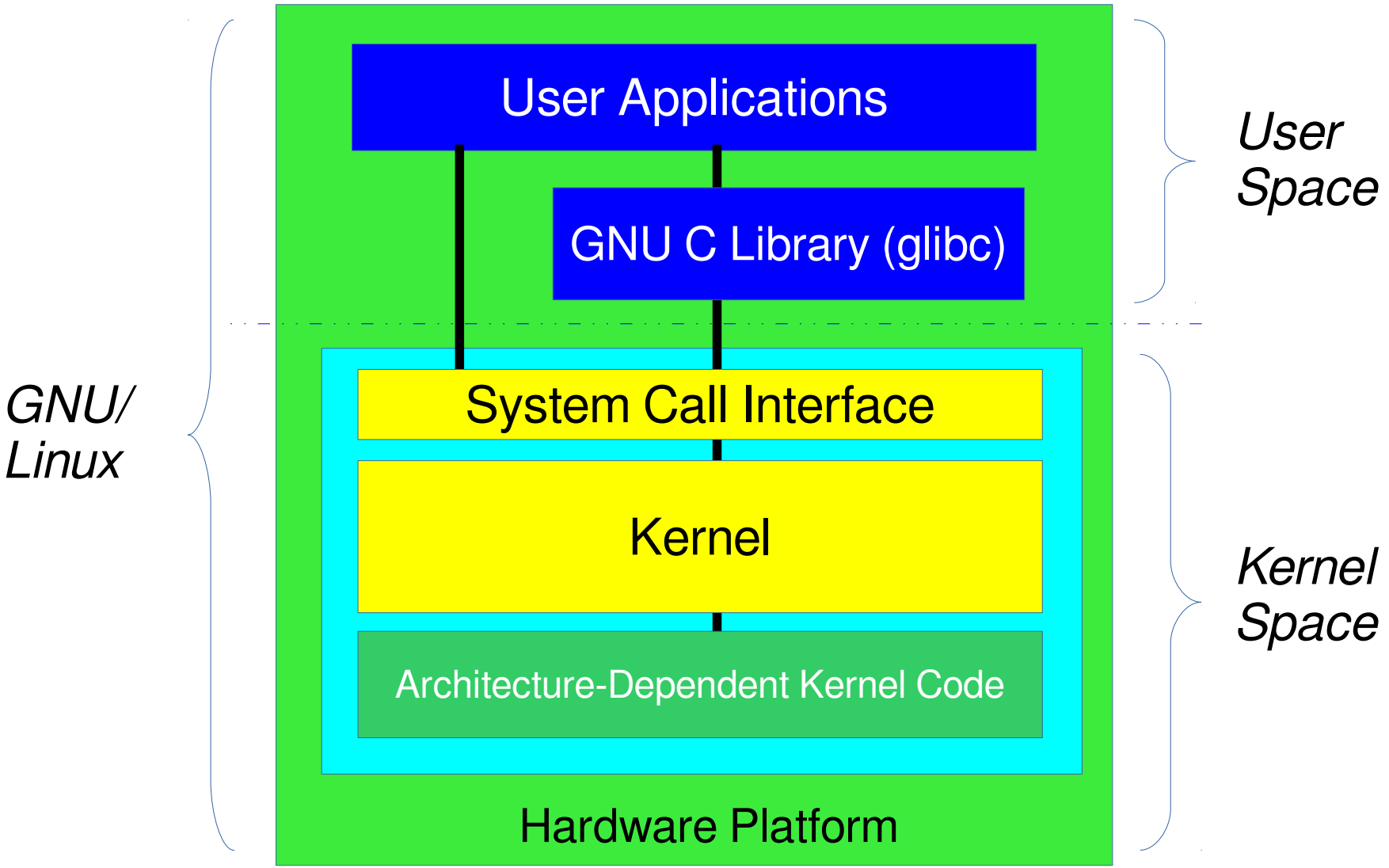
Usage Policies
(Be a good compute citizen)

- It is illegal to share computer passwords and accounts by state law and university regulation
- It is prohibited to use Ada in any manner that violates the United States export control laws and regulations, EAR & ITAR
- Abide by the expressed or implied restrictions in using commercial software

https://hprc.tamu.edu/wiki/index.php/Ada:Policies

(Brief) Introduction to Linux

Linux System Architecture



Where Are You after you login?

- ***pwd*** command (Print Current/Working Directory)

```
$ pwd
/home/user_NetID
```

Listing Files and Directories: the ***ls*** cmd

```
$ ls [options] [directory or file name]
```

- Commonly used options
  - **-l** display contents in “long” format
  - **-a** show all file (including hidden files - those beginning with .)
  - **-t** sort listing by modification time
  - **-r** reverse sort order
  - **-F** append type indicators with each entry (* / = @ |)
  - **-h** print sizes in user-friendly format (e.g. 1K, 234M, 2G)

Exercise:

```
$ ls
$ ls -a
```

```
$ touch hello.txt
$ ls
$ ls *.txt
```

Copying Files: the ***cp*** cmd

```
$ cp [options] [source] [target]
```

- If source is a file, and…
  - target is a new name: duplicate source and call it target
  - target is a directory: duplicate source, with same name, and place it in directory
- If source is a directory and the ***-r*** option is used…
  - Target is a new name: copy directory and its contents recursively into directory with new name
  - Target is a directory: duplicate source, with same name, and place it in directory

Exercise:

```
$ cp hello.txt world.txt
$ ls
```

```
$ mkdir dir1
$ cp hello.txt dir1/f1.txt
$ ls dir1
```

Moving/Renaming Files: the ***mv*** cmd

```
$ mv [source] [target]
```

- If the source is a directory name, and…
  - target is an existing dir: source dir is moved inside target dir
  - target is new name: source dir is re-named to new name
- If the source is a file name, and…
  - target is an existing dir: source file is moved inside target dir
  - target is a new name: source file is re-named to new name

Exercise:

```
$ mv hello.txt save.txt
$ ls
```

```
$ mv save.txt dir1
$ ls
$ ls dir1
```

Deleting Files: the ***rm*** cmd

```
$ rm [options] [file name]
```

- Commonly used options
  - -i prompt user before any deletion
  - -r remove the contents of directories recursively
  - -f ignore nonexistent files, never prompt

- To remove a file whose name starts with a '**–**', for example '***–foo***', use one of these commands:

```
$ rm -- -foo
$ rm ./-foo
```

Exercise:

```
$ rm world.txt
$ ls
```

```
$ rm dir1
$ rm -rf dir1
$ ls
```

Common Directory Commands

- To make a new directory:

```
$ mkdir [directory name]
```

- To change to another directory:

```
$ cd [directory name]
```

- To remove a directory (which must be empty):

```
$ rmdir [directory name]
```

Exercise:

```
$ mkdir dir2
$ touch dir2/f2.txt
$ ls
$ ls dir2
```

```
$ pwd
$ cd dir2
$ pwd
$ cd ..
$ pwd
```

```
$ rmdir dir2
$ ls dir2
$ rm dir2/f2.txt
$ rmdir dir2
$ ls
```

Changing Directories: the *cd* cmd

```
$ cd [directory name]
```

- To switch to the most recent previously used directory:

```
$ cd -
```

- To switch to the parent directory of the current directory:

```
$ cd ..
```

- Return to home directory

```
$ cd
```

or

```
$ cd ~
```

Exercise:

```
$ mkdir dir3
$ mkdir dir3/dir4
$ cd dir3
$ pwd
$ cd dir4
$ pwd
```

```
$ cd ..
$ pwd
$ cd dir4
$ pwd
$ cd -
$ pwd
```

```
$ cd
$ pwd
$ cd dir3
$ pwd
$ cd ~
$ pwd
```

Displaying File Contents

```
$ cat [file name]
```

- Short for “concatenate”, ***cat*** dumps the contents of a file (or list of files) to the screen.
- The ***more*** command, and its improved version ***less***, display an text file one page at a time.

```
$ more [file name]

$ less [file name]
```

- Other related commands:
  - ***head*****:** output the first part of files
  - ***tail*****:** output the last part of files
  - ***wc*** (word count) or ***wc -l*** (line count)

Exercise:

```
$ cat /etc/hosts
$ more /etc/hosts
$ less /etc/hosts
$ wc -l /etc/hosts
```

Documentation: the ***man*** cmd

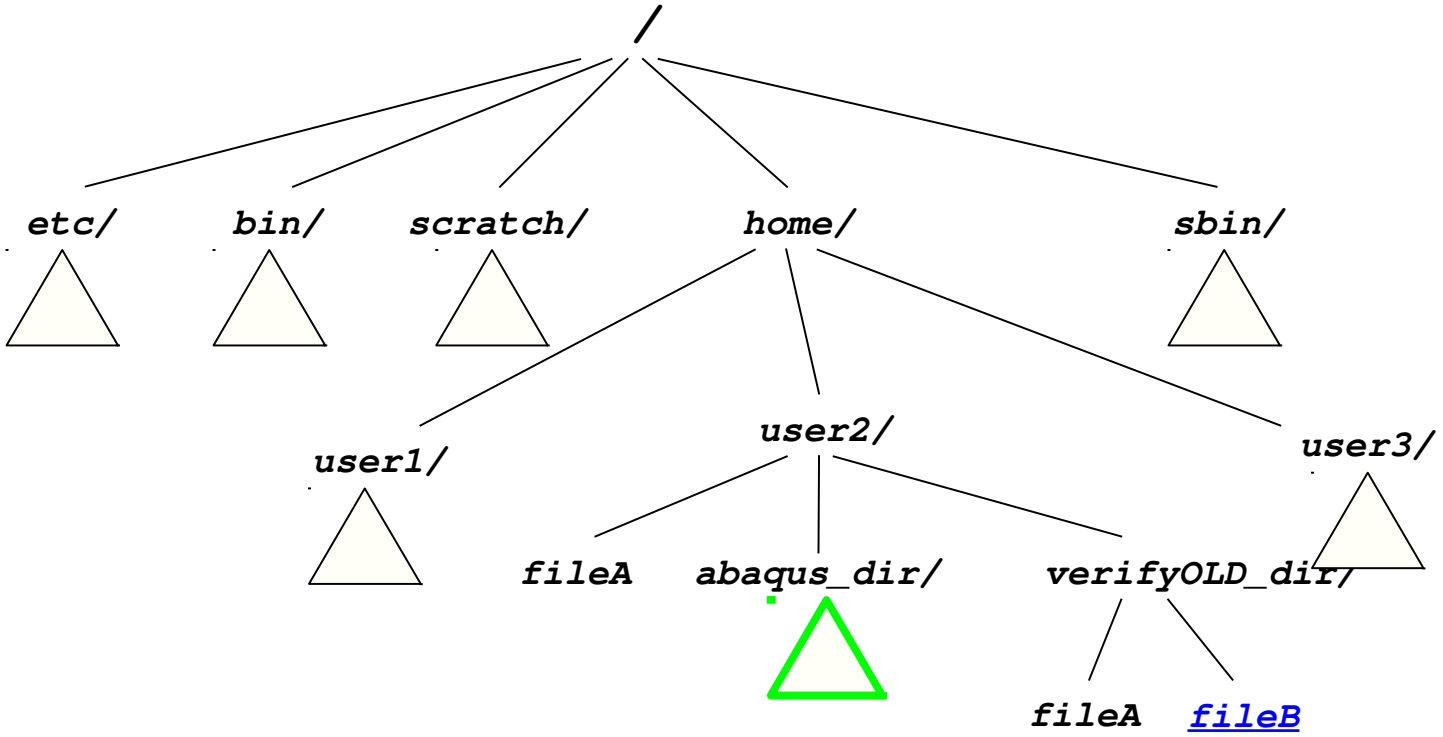
```
$ man cmd_name

$ man -k cmd_name

$ man -M <non_std_dir> cmd_name
```

- Searches in system dirs and displays associated page(s)
- The layout of a man page follows certain conventions
  - Each man page is assigned a section # (of a virtual UNIX manual database) to which it belongs;
  - Each page itself is divided into sections. The man command can search the NAME sections for keywords (-k option)
- The -M option can force man to read pages installed in non-default locations

man page layout: by sections

```
SSH(1)                    BSD General Commands Manual                    SSH(1)


NAME
     ssh - OpenSSH SSH client (remote login program)


SYNOPSIS
     ssh [-l login_name] hostname | user@hostname [command]
       :
       :
DESCRIPTION
     ssh (SSH client) is a program for logging into a remote machine and for
     executing commands on a remote machine.
       :
       :
CONFIGURATION FILES
    ssh may additionally obtain configuration data from a per-user configuration
      :
ENVIRONMENT
    ssh will normally set the following environment variables:
      :
      :
FILES
     $HOME/.ssh/known_hosts
       :
SEE ALSO
     rsh(1), scp(1), sftp(1), ssh-add(1), ssh-agent(1), ssh-keygen(1),
```

Absolute vs Relative Pathname



- For file ***fileB*** under /home/user2/verifyOLD_dir:
  - The **absolute** (*full)* pathname is: ***/home/user2/verifyOLD_dir/fileB***
  - The ***relative*** pathname is: ***verifyOLD_dir/fileB,*** if the current working directory is ***/home/user2/***

```
$ pwd
/home/user2/abaqus_dir
$ ls ../verifyOLD_dir/fileB
```

File and directory names

- Filename/directory name is a string (i.e., a sequence of characters) that is used to identify a file/directory;
- Commonly used characters: A-Z, a-z, 0-9, . (period), - (hyphen), _ (underline);
- DO NOT USE Linux/UNIX reserved characters: /, >, <, | (pipe), : (colon), & (ampersand);
- Acceptable characters but should be avoided when possible: blank spaces, ( ) parentheses, ' “ quotes (single/double), ? question mark, * asterisk, \ backslash, $ dollar sign;
- Don’t start or end your filename with a space, period, hyphen, or underline.
- Filename is case sensitive; if possible, *avoid blank space* in the file name ("*my data file*" vs "*my_data_file.txt*").

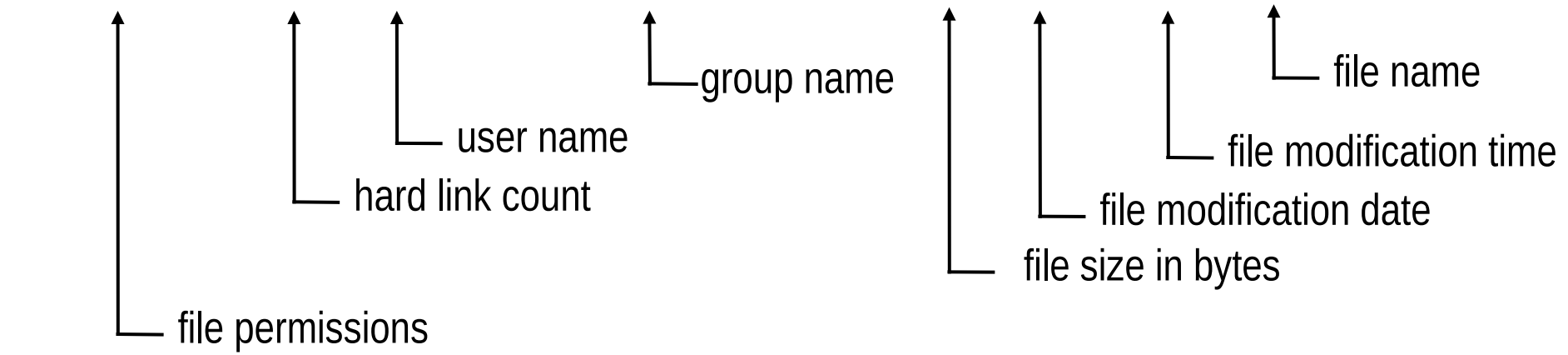
Editing an ASCII file

- There are many editors available under Linux.
- Text mode
  - nano (simple)
  - vi/vim (more advanced)
  - emacs (more advanced)
- Graphic mode (require X11)
  - gedit
  - mousepad
  - xemacs / gvim
- Be aware of text file edited under Windows (CR/LF vs LF). Use ***dos2unix*** to convert a DOS/Windows edited text file to UNIX format.

```
$ dos2unix myDOSfile.txt
```

File Attributes: A look with '*ls -l*'

```
[user_NetID@ada ~]$ ls -l
total 37216
drwx------    7 user_NetID    user_NetID         121 Sep  9 10:41 abaqus_files
-rw-------    1 user_NetID    user_NetID        2252 Aug 24 10:47 fluent-unique.txt
-rw-------    1 user_NetID    user_NetID    13393007 Aug 24 10:40 fluent-use1.txt
-rw-------    1 user_NetID    user_NetID         533 Aug 24 11:23 fluent.users
drwxr-xr-x    3 user_NetID    user_NetID          17 May  7 16:56 man
-rw-------    1 user_NetID    user_NetID    24627200 Sep  9 10:49 myHomeDir.tar
lrwxrwxrwx    1 root          root                21 May 28 16:11 README ->
/usr/local/etc/README
-rwx------    1 user_NetID    user_NetID         162 Sep  7 12:20 spiros-ex1.bash
-rwx--x--x    1 user_NetID    user_NetID          82 Aug 24 10:51 split.pl
drwxr-xr-x    2 user_NetID    user_NetID           6 May  5 11:32 verifyOLD
```



File Ownership and Permissions

```
-rwx--x--x    1 user_NetID   staff          82 Aug 24 10:51 split.pl
```

permissions

user and group ownership

Octal	Binary	Permissions
0	000	`---`
1	001	`--x`
2	010	`-w-`
3	011	`-wx`
4	100	`r--`
5	101	`r-x`
6	110	`rw-`
7	111	`rwx`

- There are 3 sets of permissions for each file
  - 1st set - user (the owner)
  - 2nd set - group (to which file owner belongs)
  - 3rd set - other (all other users)
- The **r** indicates read permission
- The **w** indicated write permission
- The **x** indicated execute permission

Directory Permissions

```
drwx------    7 user_NetID   staff         121 Sep  9 10:41 abaqus_files
```

- The meanings of the permission bits for a directory are slightly different than for regular files:
  - ***r*** permission means the user can list the directory's contents
  - ***w*** permission means the user can add or delete files from the directory
  - ***x*** permission means the user can cd into the directory; it also means the user can execute programs stored in it
- Notice that if the file is a directory, the leading bit before the permissions is set to ***d*** , indicating directory.

Changing Attributes: the ***chmod*** cmd

```
$ chmod [options] [permission mode] [target_file]
```

```
$ chmod 777 myFile.txt    ( the permissions will be set to rwxrwxrwx )

$ chmod o-x myFile.txt    ( the permissions will change to rwxrwxrw- )

$ chmod gu-x myFile.txt   ( the permissions will change to rw-rw-rw- )

$ chmod u+x myFile.txt    ( the permitions will change to rwxrw-rw- )
```

-R is a commonly used option. It recursively applies the specified permissions to all files and directories within target, if target is a directory.

References

Extended version of Introduction to Linux can be found at

http://hprc.tamu.edu/shortcourses/SC-unix/

Here are some slides from TACC and LSU on the similar subject.

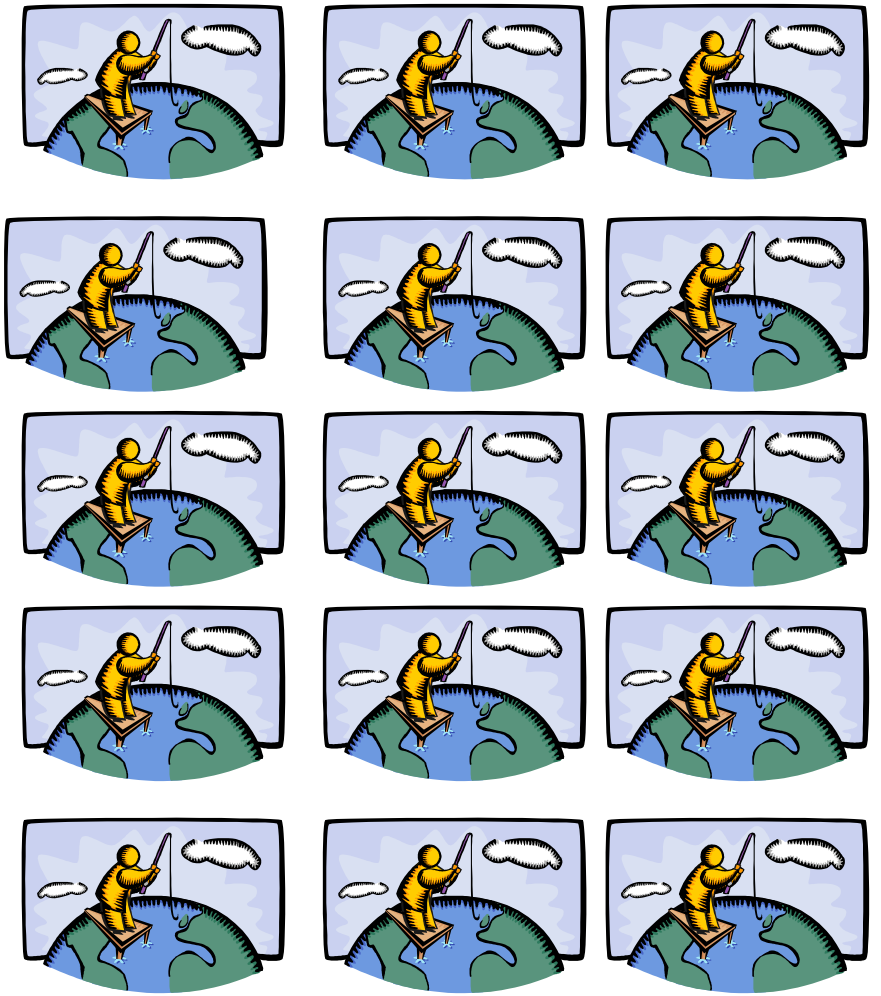
- Linux/Unix Basics for HPC: October 9, 2014 (with video) [TACC]
  https://portal.tacc.utexas.edu/-/linux-unix-basics-for-hpc
- Express Linux Tutorial: Learn Basic Commands in an Hour [TACC]
  https://portal.tacc.utexas.edu/c/document_library/get_file?uuid=ed6c16e9-bcbc-4b70-9311-5273b09508b8&groupId=13601
- Introduction to Linux for HPC [LSU]
  http://www.hpc.lsu.edu/training/weekly-materials/2015-Fall/intro-linux-2015-09-02.pdf

Introduction to Using the Ada Cluster

Parallelism

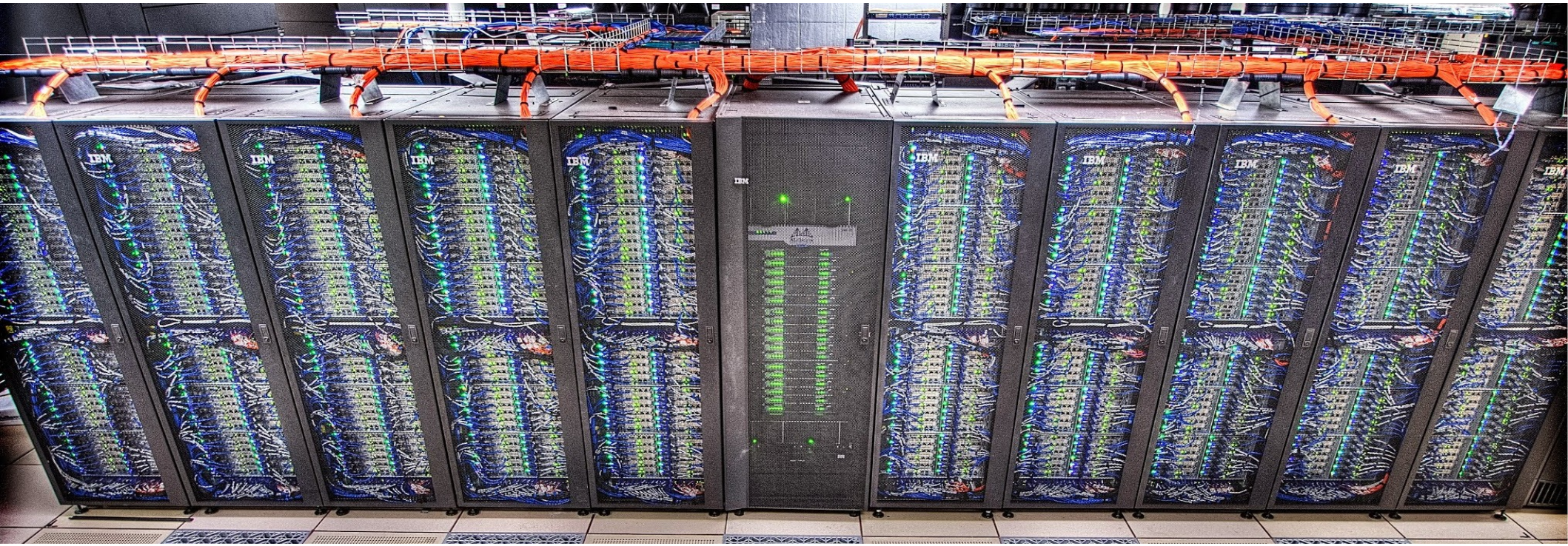
Parallelism means doing multiple things at the same time: you can get more work done in the same time.

Less fish …



More fish!

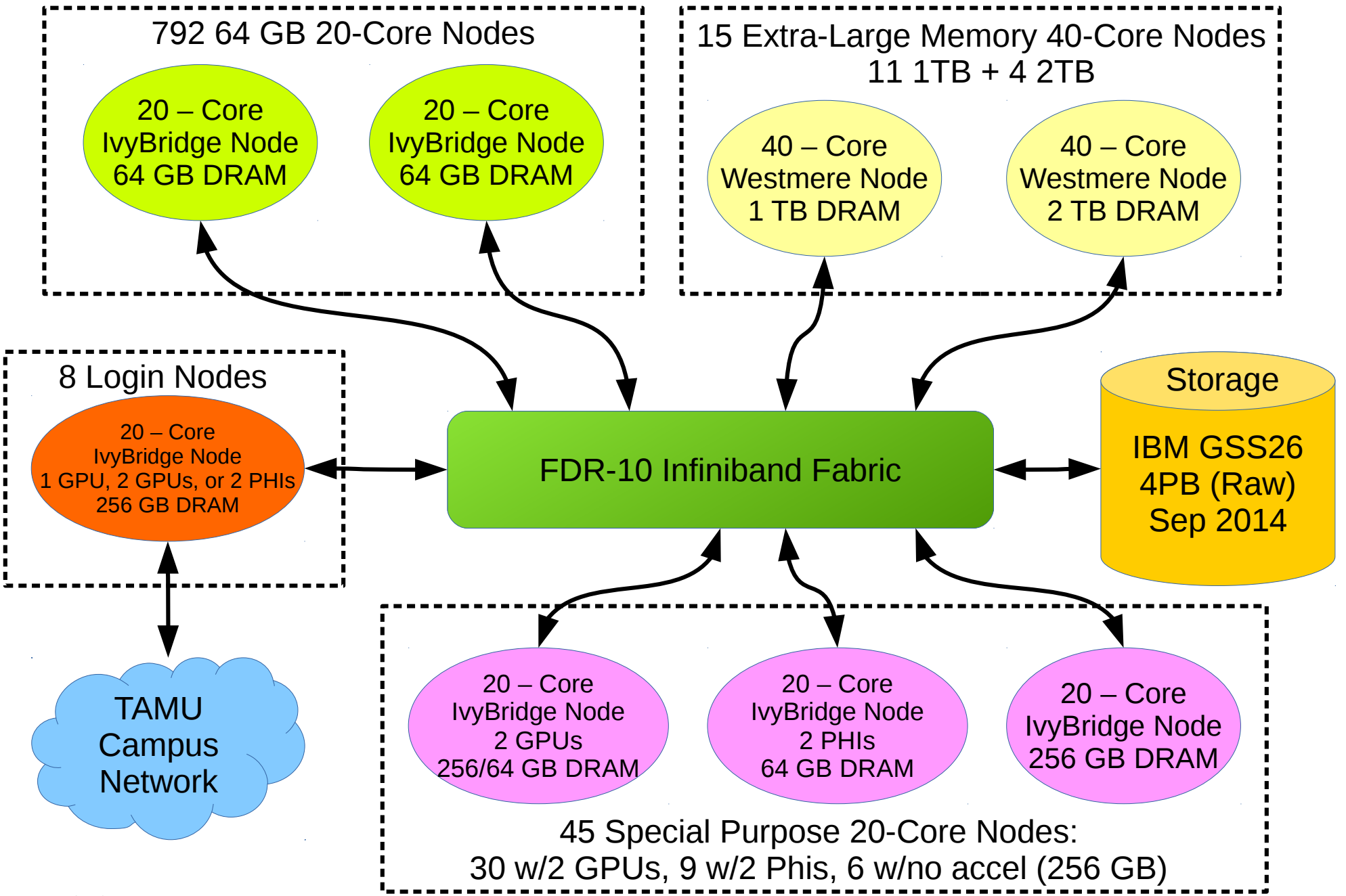
Ada – an x86 Cluster



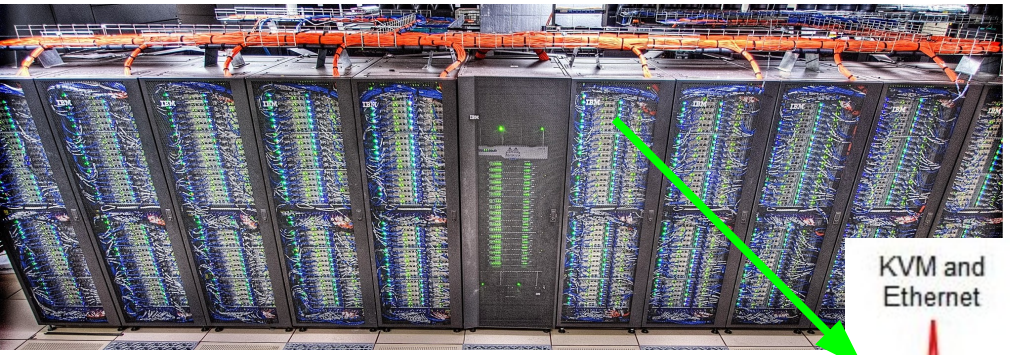
A 17,500-core, 860-node cluster with:

- 837 20-core compute nodes with two Intel 10-core 2.5GHz IvyBridge processors.
  - Among these nodes, 30 nodes have 2 GPUs each and 9 nodes have 2 Phi coprocessors.
- 15 compute nodes are 1TB and 2TB memory, 4-processor SMPs with the Intel 10-core 2.26GHz Westmere processor.
- 8 20-core login nodes with two Intel 10-core 2.5GHz IvyBridge processors and 1 GPU, 2 GPUs, or 2 Phi coprocessors
- Nodes are interconnected with FDR-10 InfiniBand fabric in a two-level (core switch shown above in middle rack and leaf switches in each compute rack) fat-tree topology.

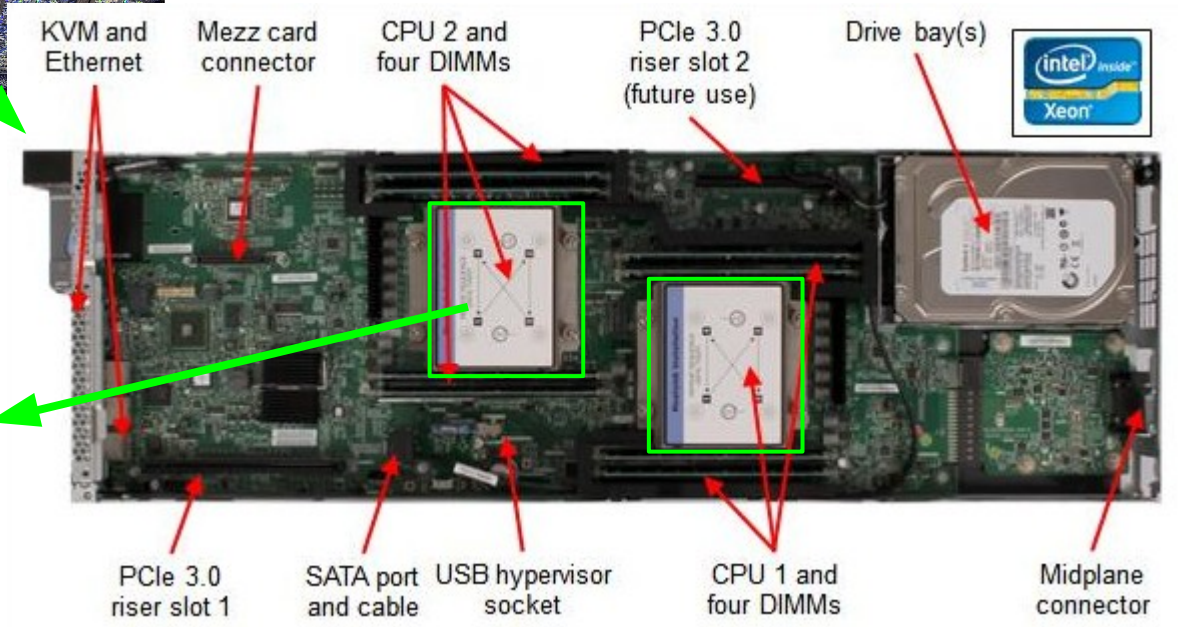
Ada Schematic: 17,500-core 860-node Cluster



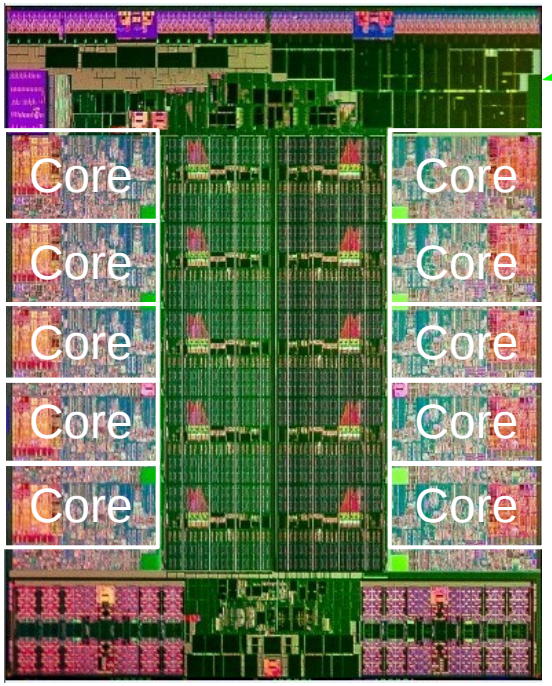
Node / Socket / Core



Part of Ada cluster.
Each blue light is a node.



Each node has 2 sockets.



Each socket/CPU has 10 processor cores.
So, each node has 20 processor cores.

Accessing Ada

- SSH is required for accessing Ada:
  - On campus: ssh NetID@ada.tamu.edu
  - Off campus:
    - Set up VPN: http://hdc.tamu.edu/Connecting/VPN/
    - Then: ssh NetID@ada.tamu.edu
- SSH programs for Windows:
  - MobaXTerm (preferred, includes SSH and X11)
  - PuTTY SSH
- Login sessions that are idle for 60 minutes will be closed automatically

https://hprc.tamu.edu/wiki/index.php/HPRC:Access

File Transfers with Ada

- Simple File Transfers:
  - scp: command line   (Linux, MacOS)
  - WinSCP: GUI (Windows)
  - FileZilla: GUI (Windows, MacOS, Linux)
- Bulk data transfers:
  - Use fast transfer nodes (FTN) with:
    - GridFTP
    - Globus Connect
    - bbcp

https://hprc.tamu.edu/wiki/index.php/Ada:Filesystems_and_Files#Transfering_Files

File Systems and User Directories

Directory	Environment Variable	Space Limit	File Limit	Intended Use
/home/$USER	$HOME	10 GB	10,000	Small to modest amounts of processing.
/scratch/user/$USER	$SCRATCH	1 TB	50,000	Temporary storage of large files for on-going computations. Not intended to be a long-term storage area.
/tiered/user/$USER	$ARCHIVE	10 TB	50,000	Intended to hold valuable data files that are not frequently used

- View usage and quota limits: the ***showquota*** command
- Also, only home directories are backed up daily.
- Quota increases will only be considered for scratch and tiered directories

https://hprc.tamu.edu/wiki/index.php/Ada:Filesystems_and_Files

Computing Environment

- Paths:
  - $PATH: for commands (eg. /bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/netid/bin)
  - $LD_LIBRARY_PATH: for libraries

Try "`echo $PATH`"

- Many applications, many versions, and many paths ..... How do you manage all this software?!
- Each version of an application, library, etc. is available as a module.
- Module names have the format of package_name/version.
- Avoid loading modules in *.bashrc*

https://hprc.tamu.edu/wiki/index.php/Ada:Computing_Environment#Modules

Application Modules

- Installed applications are available as modules which are available to all users
- **`module`** commands
  - **`module avail`** `#show all available modules`
  - **`module spider`** `tool_name` `#search all modules`
  - **`module key`** `genomics` `#search with keyword`
  - **`module load`** `tool_name` `#load a specific module`
  - **`module list`** `#list loaded modules`
  - **`module purge`** `#unload all loaded modules`
  - **`module load`** `Stacks` `#load the default version of a tool`
  - **`module load`** `Stacks/1.37-intel-2015B` `#load a specific version`

Toolchains

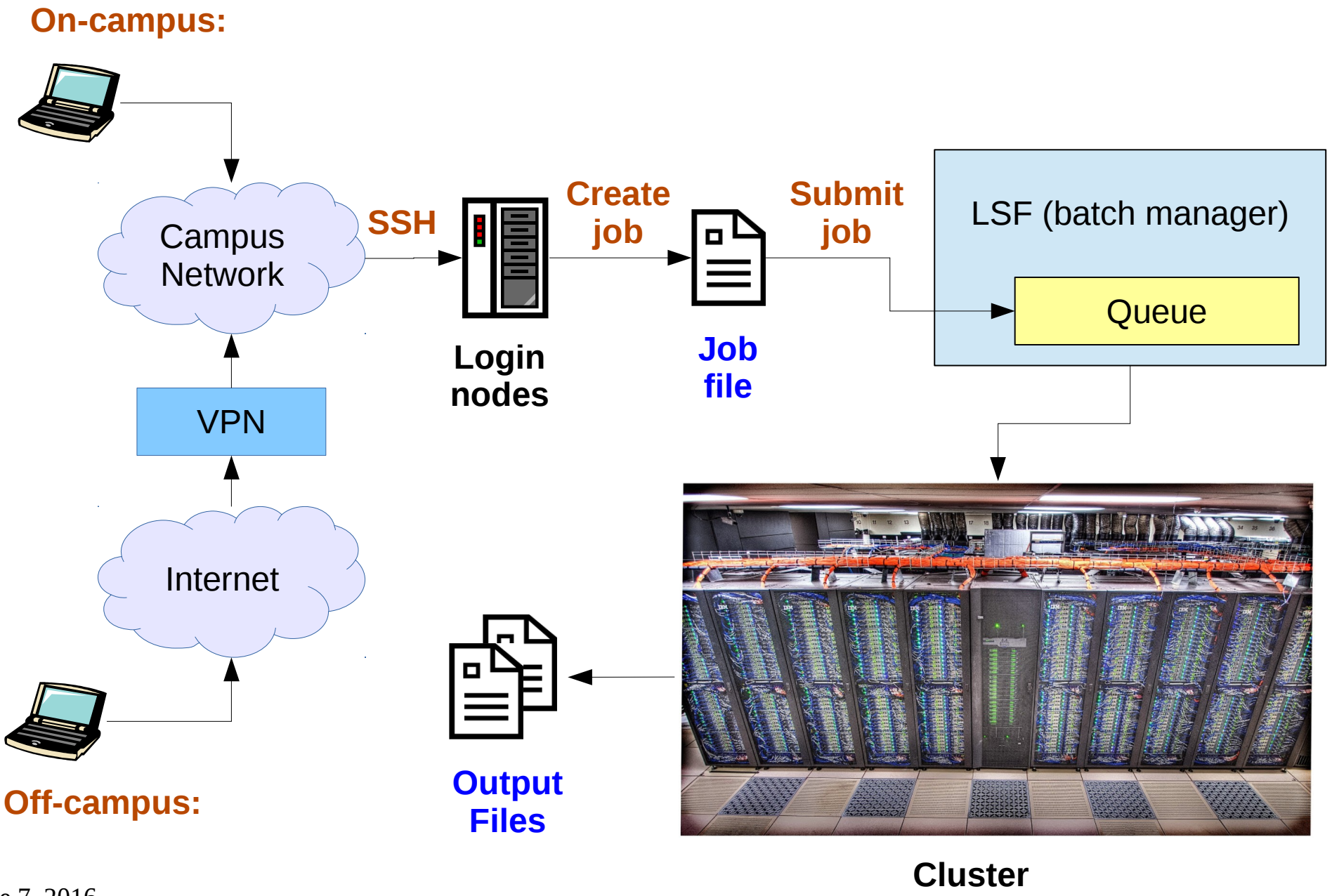
- Use the same toolchains in your job scripts
  - The **`intel-2015B`** is the recommended toolchain

```
module load Bowtie2/2.2.6-intel-2015B
module load TopHat/2.1.0-intel-2015B
module load Cufflinks/2.2.1-intel-2015B
```

- Avoid mixing tool chains if loading multiple modules in the same job script:

```
module load Bowtie2/2.2.2-ictce-6.3.5
module load TopHat/2.0.14-goolf-1.7.20
module load Cufflinks/2.2.1-intel-2015B
```

Batch Computing on Ada



Batch Queues

- Job submissions are assigned to batch queues based on the resources requested (number of cores/nodes and wall-clock limit)
- Some jobs can be directly submitted to a queue:
  - If the 1TB or 2TB nodes are needed, use the *xlarge* queue
  - Jobs that have special resource requirements are scheduled in the special queue (must request access to use this queue)
- Batch queue policies are used to manage the workload and may be adjusted periodically.

https://hprc.tamu.edu/wiki/index.php/Ada:Batch_Queues

Current Queues

$ bqueues

QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
staff	450	Open:Active	-	-	-	-	0	0	0	0
special	400	Open:Active	-	-	-	-	3044	0	3044	0
xlarge	100	Open:Active	-	-	-	-	40	0	40	0
vnc	90	Open:Active	-	-	-	-	11	10	1	0
sn_short	80	Open:Active	-	-	-	-	0	0	0	0
mn_short	80	Open:Active	2000	-	-	-	80	80	0	0
mn_large	80	Open:Active	5000	-	-	-	7960	3000	4960	0
general	50	Closed:Inact	0	-	-	-	0	0	0	0
sn_regular	50	Open:Active	-	-	-	-	8588	6220	2368	0
sn_long	50	Open:Active	-	-	-	-	2610	240	2370	0
sn_xlong	50	Open:Active	-	-	-	-	111	0	111	0
mn_small	50	Open:Active	6000	-	-	-	2232	440	1792	0
mn_medium	50	Open:Active	6000	-	-	-	800	200	600	0
curie_devel	40	Open:Active	32	32	-	-	18	0	18	0
curie_medium	35	Open:Active	512	192	-	-	1904	1456	448	0
curie_long	30	Open:Active	192	64	-	-	1264	1088	176	0
curie_general	25	Closed:Inact	0	-	-	-	0	0	0	0
preempt_medium	10	Open:Active	-	-	-	-	0	0	0	0
low_priority	1	Open:Active	2500	500	-	-	0	0	0	0
preempt_low	1	Open:Active	40	-	-	-	0	0	0	0

https://hprc.tamu.edu/wiki/index.php/Ada:Batch_Queues

Consumable Computing Resources

- Resources specified in a job file:
  - Processor cores
  - Memory
  - Wall time
  - GPU
- Other resources:
  - Software license/token
- Billing Account

Sample Job Script

```
#BSUB -L /bin/bash
#BSUB -J blastx
#BSUB -n 2
#BSUB -R "span[ptile=2]"
#BSUB -R "rusage[mem=1000]"
#BSUB -M 1000
#BSUB -W 2:00
#BSUB -o stdout.%J
#BSUB -e stderr.%J

module load BLAST+/2.2.31-intel-2015B-Python-3.4.3

<<README
    BLAST manual: http://www.ncbi.nlm.nih.gov/books/NBK279690/
README

#blastx: search protein databases using a translated nucleotide query

blastx -query mrna_seqs_nt.fasta -db /scratch/datasets/blast/nr \
-outfmt 10 -out mrna_seqs_nt_blastout.csv
```

Sample Job Script

```
#BSUB -L /bin/bash
#BSUB -J blastx
#BSUB -n 2
#BSUB -R "span[ptile=2]"
#BSUB -R "rusage[mem=1000]"
#BSUB -M 1000
#BSUB -W 2:00
#BSUB -o stdout.%J
#BSUB -e stderr.%J

module load BLAST+/2.2.31-intel-2015B-Python-3.4.3

<<README
    BLAST manual: http://www.ncbi.nlm.nih.gov/books/NBK279690/
README

#blastx: search protein databases using a translated nucleotide query

blastx -query mrna_seqs_nt.fasta -db /scratch/datasets/blast/nr \
-outfmt 10 -out mrna_seqs_nt_blastout.csv
```

These parameters are read by the job scheduler

Load the required module(s) first

This is a section of comments

This is a single line comment and not run as part of the script

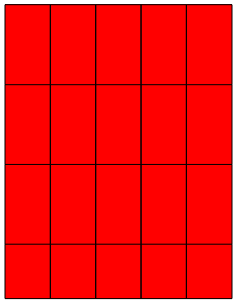
This is the command to run the application

This means the command is continued on the next line;
The space before the \ is required
Do not put a space after the \

Important Job Parameters

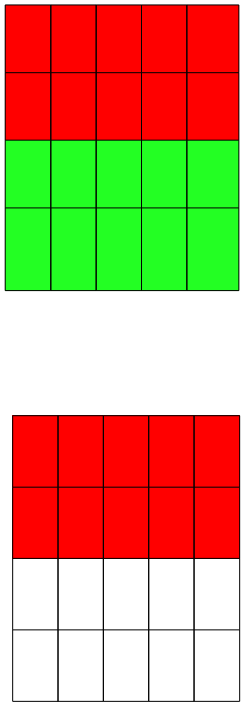
```
#BSUB -n NNN
      # NNN: total number of cores or job slots to allocate for the job
#BSUB -R "span[ptile=XX]"
      # XX:  number of cores or job slots per node to use
#BSUB -R "select[node-type]"
      # node-type: nxt, mem256gb, gpu, phi, mem1tb, mem2tb ...
#BSUB -R "rusage[mem=nnn]"
      # reserves nnn MBs per core or job slot for the job
#BSUB -M nnn
      # enforces (XX cores * nnn MB) as memory limit
      # per node for the job
#BSUB -W hh:mm or mm
      # sets job's runtime wall-clock limit in hours:minutes or just
minutes
```

Processor Cores Mapping



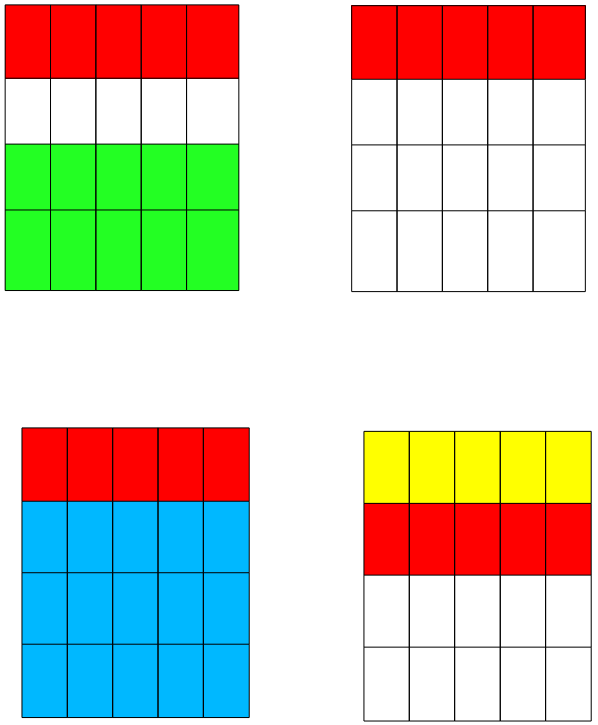
20 cores on
1 node

#BSUB -n 20
#BSUB -R “span[ptile=20]



20 cores on
2 nodes

#BSUB -n 20
#BSUB -R “span[ptile=10]



20 cores on
4 nodes

#BSUB -n 20
#BSUB -R “span[ptile=5]

Job Memory Requests

- Must specify both parameters for requesting memory:

  *#BSUB **-R** "rusage[mem=process_alloc_size]"*

  *#BSUB **-M** process_size_limit*

- Default value of 2.5 GB per job slot if -R/-M not specified, but it might cause memory contention when sharing a node with other jobs.
- On 64GB nodes, usable memory is at most 54 GB. The per-process memory limit should not exceed 2700 MB for a 20-core job.
- If more memory is needed, request the large memory nodes:
  - If under 256 GB and up to 20 cores per node: use *-R "select[mem256gb]"*
  - If need up to 1 or 2 TB of memory or up to 40 cores:
    - use *-R "select[mem1tb]"* (40 cores) or *-R "select[mem2tb]"* with the *-q xlarge* option
    - The mem1tb and mem2tb nodes are accessible only via the *xlarge* queue.

Job Submission and Tracking

Command	Description
bsub < jobfile1	Submit jobfile1 to batch system
bjobs [-u all or user_name] [[-l] job_id]	List jobs
bpeek [-f] job_id	View job's output and error files
bkill job_id	Kill a job
bhist [-l] job_id	Show historical information about a job
lnu [-l] -j job_id	Show resource usage for a job
blimits -w	Show how policies are applied to users and queues

https://hprc.tamu.edu/wiki/index.php/Ada:Batch#Job_tracking_and_control_commands

Job Environment Variables

- $LSB_JOBID = job id
- $LS_SUBCWD = directory where job was submitted from
- $SCRATCH = /scratch/user/NetID
- $TMPDIR = /work/$LSB_JOBID.tmpdir
  - $TMPDIR is local to each assigned compute node for the job

https://hprc.tamu.edu/wiki/index.php/Ada:Batch#Environment_Variables

Submit the Job and Check Status

- Submit your job to the job scheduler

```
bsub < sample01.job
```

```
Verifying job submission parameters...
Verifying project account...
     Account to charge:   082792010838
         Balance (SUs):      4871.5983
         SUs to charge:         0.0333
Job <2470599> is submitted to default queue <sn_short>.
```

- Summary of the status of your running/pending jobs

```
bjobs
```

```
JOBID   STAT USER       QUEUE    JOB_NAME  NEXEC_HOST SLOTS RUN_TIME     TIME_LEFT
2470599 RUN  tmarkhuang sn_short sample01  1          1     0 second(s)  0:5 L
```

- A more detailed summary of a running job

```
bjobs -l 2470599
```

Try yourself: *cp -r /scratch/training/Bioinformatics_Workshop $SCRATCH/*

Debug job failures

- Debug job failures using the stdout and stderr files
  - `cat output.ex03.python_mem.2447336`

This job id was created by the parameter in your job script file
`#BSUB -o output.ex03.python_mem.%J`

```
TERM_MEMLIMIT: job killed after reaching LSF memory usage limit.
Exited with signal termination: Killed.

Resource usage summary:

    CPU time :                                   1.42 sec.
    Max Memory :                                 10 MB
    Average Memory :                             6.50 MB
    Total Requested Memory :                     10.00 MB
    Delta Memory :                               0.00 MB
    Max Processes :                              5
    Max Threads :                                6
```

Make the necessary adjustments to BSUB parameters in your job script and resubmit the job

Check your Service Unit (SU) Balance

- Show the SU Balance of your Account(s)

```
myproject -l
```

- Use "**#BSUB -P project_id**" to charge SU to a specific project
- Back up your job scripts often
  - Submitting a job using > instead of < will erase the job script content

Common Job Problems

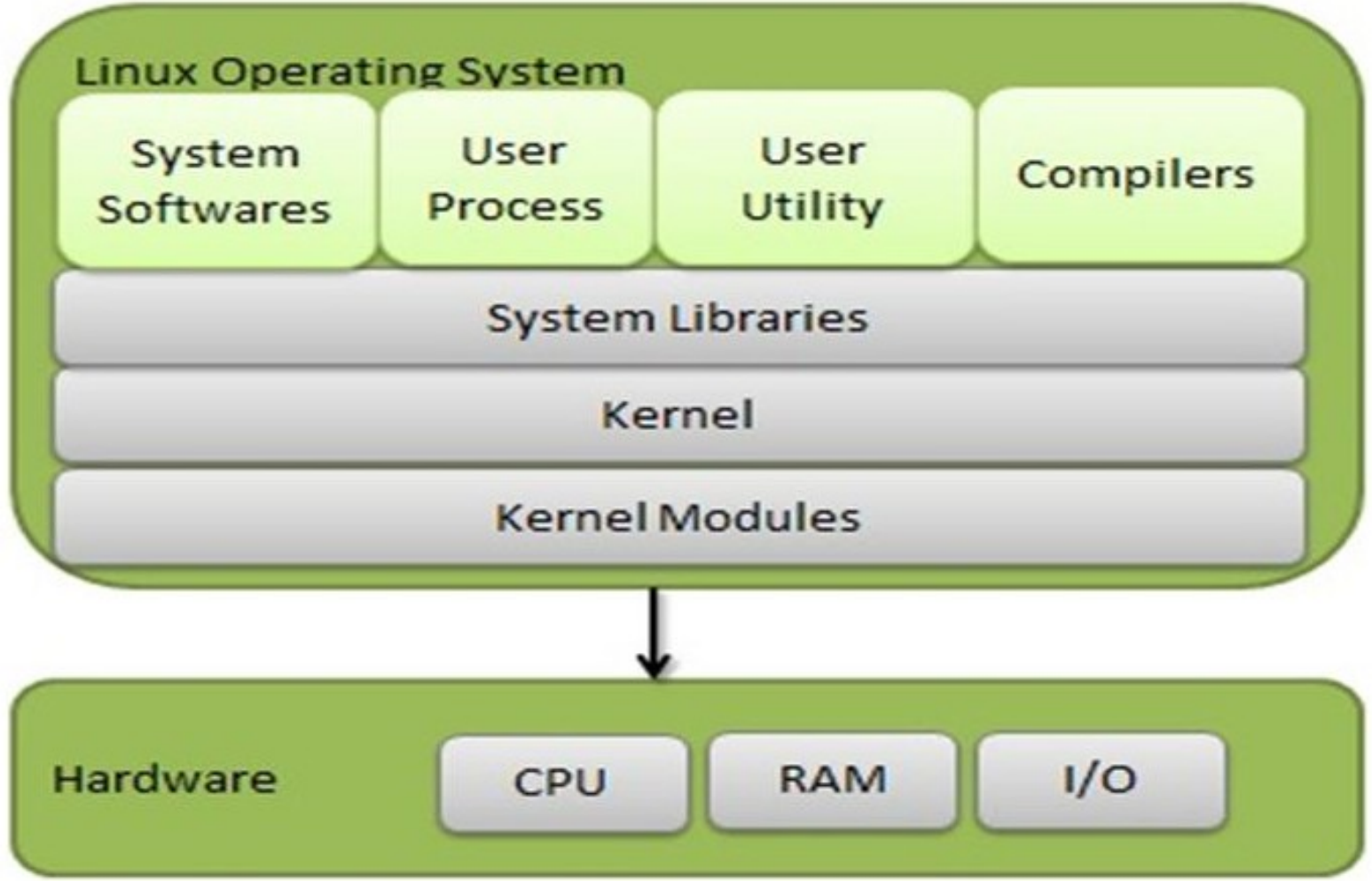
- Control characters (**^M**) in job files or data files edited with DOS/Windows editor
  - remove the **^M** characters with: `dos2unix my_job_file`
- Did not load the required module(s)
- Insufficient walltime specified in #BSUB -W parameter
- Insufficient memory specified in #BSUB -M and -R "rusage[mem=xxx]" parameters
- No matching resource (-R rusage[mem] too large)
- Running OpenMP jobs across nodes
- Insufficient SU: See your SU balance: `myproject -l`
- Insufficient disk or file quotas: check quota with `showquota`
- Using GUI-based software without setting up X11 forwarding
  - Enable X11 forwarding at login `ssh -X user@ada.tamu.edu`
- Software license availability

Need Help?

- Check the Ada User Guide (https://hprc.tamu.edu/wiki/index.php/Ada) for possible solutions first.
- Email your questions to **help@hprc.tamu.edu**. (Now managed by a ticketing system)
- Help us, help you -- we need more info
  - Which Cluster
  - UserID/NetID (*UIN is not needed!*)
  - Job id(s) if any
  - Location of your jobfile, input/output files
  - Application used if any
  - Module(s) loaded if any
  - Error messages
  - Steps you have taken, so we can reproduce the problem
- Or visit us @ TAES 103
  - Making an appointment is recommended.

Backup Slides

Linux System Architecture



Source: http://googler700.blogspot.com/2015/03/operating-system-linux.html

Types of File: the ***file*** cmd

```
$ file [name]
```

- Displays a brief description of the contents or other type information for a file or related object.

```
$ file hello.c
hello.c: ASCII C program text
```

The UNIX Filesystem

What is a Computer Filesystem?

- A software platform/system which *provides the tools for the way storage space is organized on mass storage media* in terms of different file objects of all sorts. It provides for, among other, the creation, access, and modification of these file objects;
- The term also refers to organized space-on some medium-as carried out by a computer file system;
- UNIX-based file systems organize storage space in specific ways;
- Many computer file systems are available: EXT3/4, XFS, JFS, ZFS, etc.

The ***tree*** cmd

```
$ tree [dir_name]
```

- Shows the contents of a directory structure in a hierarchical arrangement.

```
$ tree bin
bin
├── perlsh
└── xtail.pl

0 directories, 2 files
```

The '*find*' Command

```
$ find [target dir] [expression]
```

```
$ find . -name "*.txt" -print

$ find . -newer results4.dat -name "*.dat" -print

$ find /scratch/user_NetID -mtime +2  -print

$ find /scratch/user_NetID -mtime -7  -print

$ find /tmp -user user_NetID -print
```

Comparing files – '***diff*** and '***cmp***'

```
$ diff [options] FILES

# basic example

   $ diff file1 file2

# side by side comparison (long line truncated):

   $ diff -y file1 file2

# side by side comparison with screen width of 180 characters

   $ diff -y -W 180 file1 file2
```

```
$ cmp file1 file2
```

'***grep***' – Search pattern(s) in files

```
$ grep [options] PATTERN [FILES ...]
```

basic example

```
   $ grep GoodData mydata.txt
```

search multiple matches

```
   $ grep -e GoodData -e Important mydata.txt
```

excluding a pattern; show non-matched lines

```
   $ grep -v NG mydata.txt
```

```
$ cat mydata.txt | grep GoodData
$ grep -v junk mydata.txt | grep -v NG
$ grep -e "^OUTPUT" mydata.txt
```

The '***tar***' Command

```
$ tar [options] [tar file] [file or dir name]
```

- Used to "package" multiple files (along with directories if any) into one file suffixed with a ***.tar*** suffix by convention.
- Commonly used options
  - **x** extract files from a tar
  - **c** create a new tar
  - **t** list the contents of a tar
  - **v** verbosely list files processed
  - **f** use the specified tar file
  - **z** the tar file is compressed

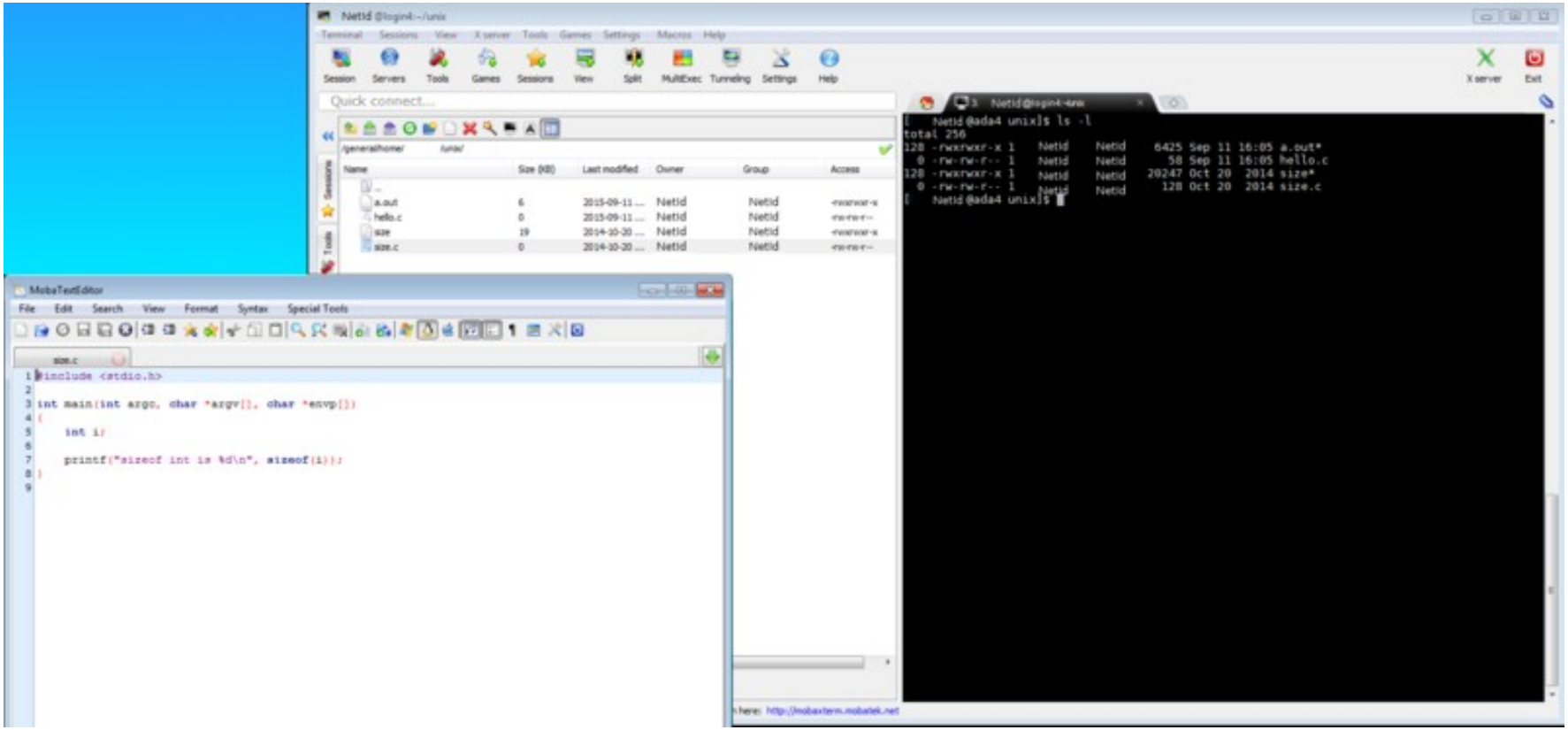
The '***tar***' Command - examples

```
$ tar cvf myHomeDir.tar .
        (package the current dir into a file called myHomeDir.tar)
$ tar tvzf Interesting.tar.gz
        (show the content for the compressed tar file)
$ tar xvfz RunLogs.tgz ./abaqus_files
        (extract the dir abaqus_files from the compressed tar file)
```

- Be careful when extracting files (overwriting old files).
- Where files are extracted depends on how they were packaged.
- Always a good idea to check Table of Contents (**`-t`** option) before extraction.

Transfer data between Windows hosts with MobaXterm

- On a Windows system, you can use MobaXterm to transfer files to/from a server

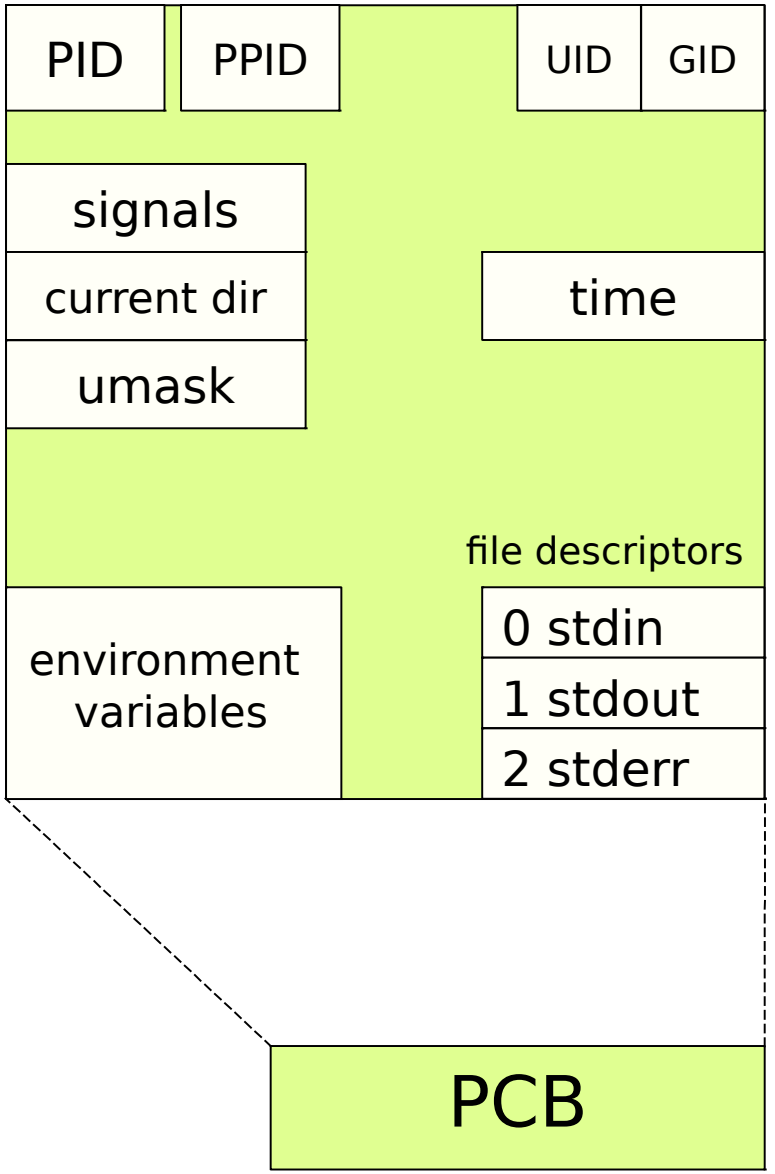


Ref: https://sc.tamu.edu/wiki/index.php/TAMUSC:MobaXterm#File_Transfers

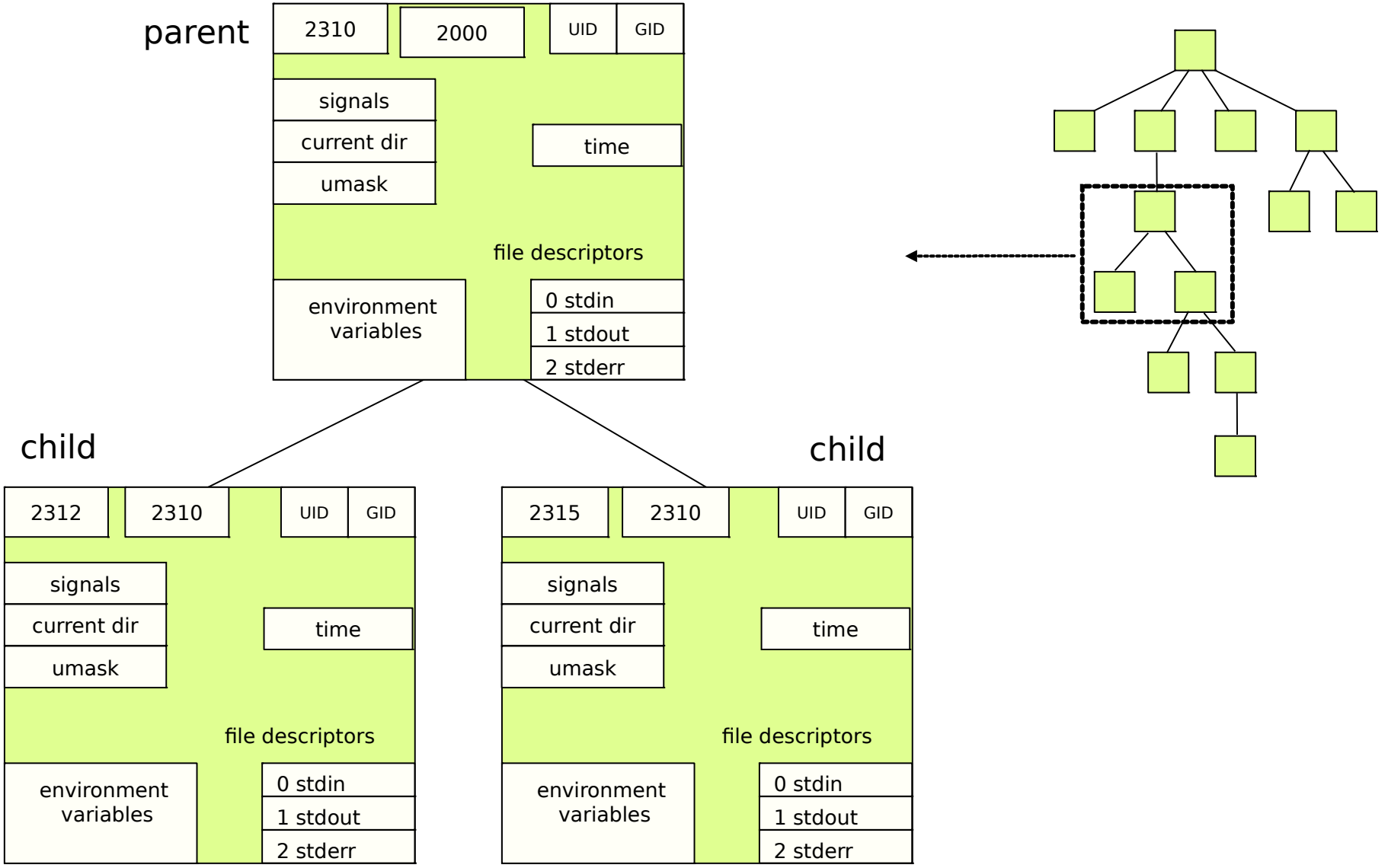
UNIX Processes

Process Attributes

- **PID:** each process is uniquely identified by an integer (process ID), assigned by the kernel
- **PPID:** parent process ID
- **UID:** integer ID of the user to whom the process belongs
- **GID:** integer ID of the user group to which the process belongs
- **Signals:** how this process is configured to respond to various signals (more later…)
- **Current dir:** the current working directory of the process
- **Environment variables:** variables that customize the behavior of the process (e.g. TERM)
- **File descriptors:** a table of small unsigned integers, starting from 0, that reference open "files" used by the process (for receiving input or producing output).



The Process Hierarchy



Shell Variables

- There are two types of variables: *local* and *environment*
  - ***Local***: known only to the shell in which they are created
  - ***Environment***: available to any child processes spawned from the shell from which they were created
  - Some variables are user-created, other are special shell variables (e.g. PWD)
- Variable name must begin with an alphabetic or underscore (_) character. The remaining characters can be alphabetic, numeric, or the underscore (any other characters mark the end of the variable name).
  - Names are case-sensitive
  - When assigning values, no whitespace surrounding the = sign
  - To assign null value, follow = sign with a newline

Some Special Variables

- ***$$*** The PID of the current shell process
- ***$-*** The sh options currently set
- ***$?*** The exit value of the last executed command
- ***$!*** The PID of the last job put in the background

```
$ echo $$
6125
$ echo $-
himBH
$ echo $?
0
$ echo $!

$
```

Environment Variables

- The ***export*** command makes variables available to child processes.

```
$ export NAME=”user_NetID Jackson”
$ MYNAME=user_NetID
$ export MYNAME
```

- The scope of these environment variables is not limited to the current shell.
- Some environment variables, such as HOME, LOGNAME, PATH, and SHELL are set by the system as the user logs in.
- Various environment variables are often defined in the *.bash_profile* startup file in the user’s home directory.
- The ***export –p*** command lists all environment (or global) variables currently defined.

Bash Environment Variables

HOME	pathname of current user's home directory
PATH	the search path for commands. It is a colon separated list of directories in which the shell looks for commands.
PPID	PID of the parent process of the current shell
PS1	primary prompt string ('$' by default)
PWD	present working directory (set by cd)
SHELL	the pathname of the current shell
USER	the username of the current user

```
$ echo $HOME
/home/user2
```

The Search Path

- The shell uses the PATH environment variable to locate commands typed at the command line
- The value of PATH is a colon separated list of full directory names.
- The PATH is searched from left to right. If the command is not found in any of the listed directories, the shell returns an error message
- If multiple commands with the same name exist in more than one location, the first instance found according to the PATH variable will be executed.

```
PATH=/opt/TurboVNC/bin:/software/tamusc/local/bin:/software/lsf/9.1/linux2.6-glibc2.3-
x86_64/etc:/software/lsf/9.1/linux2.6-glibc2.3-
x86_64/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/usr/lpp/mmfs/bin
:/opt/ibutils/bin:/home/user_NetID/bin
```

Unsetting Variables

- Both local and environment variables can be unset by the ***unset*** command.

```
$ unset name
$ unset TERM
```

- Only those variables defined as read-only cannot be unset.

Other useful shell tips

- TAB-completion: Use TAB key to automatic completion when typing file, directory or command name
- Reverse history search: Use "^R" (Ctrl-R) to perform "reverse-i-search" in command history
- Use '***`history`***' to show command history; "***`!!`***" for previous command; more on this later

Process Signals

What is a Process?

- Process
  - A program that is loaded into memory and executed
- Program
  - Machine readable code (binary) that is stored on disk
- The kernel (OS) controls and manages processes
  - It allows multiple processes to share the CPU (multi-tasking)
  - Manages resources (e.g. memory, I/O)
  - Assigns priorities to competing processes
  - Facilitates communication between processes
  - Can terminate (kill) processes

The '*ps*' Command

```
$ ps [options]
```

- Commonly used options (ps can take different styles of options)
  - **a** select all processes on a terminal (including those of other users)
  - **u** associate processes w/ users in the output
  - **f** ASCII-art process hierarchy (forest)
  - **-l** display in long format

See man page for '*ps*' for more options. (Run "*man ps*")

Output of the '*ps*' Command

```
[user_NetID@ada ~]$ ps
  PID TTY          TIME CMD
27689 pts/8    00:00:00 bash
28023 pts/8    00:00:00 ps


[user_NetID@ada ~]$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1795 27689 27688  0  80   0 - 27117 wait   pts/8    00:00:00 bash
0 R  1795 28011 27689  0  80   0 - 27035 -      pts/8    00:00:00 ps


[user_NetID@ada ~]$ ps uf
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
kjackson 27689  0.0  0.0 108468  1988 pts/8    Ss   11:14   0:00 -bash
kjackson 28043  1.0  0.0 110208  1052 pts/8    R+   11:22   0:00  \_ ps uf
```

The '***kill***' Command

```
$ kill -l
$ kill [signal name] pid
```

- The ***kill -l*** command lists all the signal names available.
- The ***kill*** command can generate a signal of any type to be sent to the process specified by a PID.
- The ***kill -9*** sends the (un-interruptable) kill signal.
- 'kill' is the not the best name for this command, because it can actually generate any type of signal.

Job Control

- Job control is a feature of the bash sell that allows users to manage multiple simultaneous processes launched from the same shell.
- With job control, one can send a job running in the foreground to the background, and vice versa.
- Job control commands:

jobs	lists all the jobs running
^z (ctrl-z)	stops (suspends) the job running in the foreground
bg	starts running the stopped job in the background
fg	brings a background job to the foreground
kill	sends a kill signal to a specified job
cmd ***&***	execute *cmd* in the background ($!)
wait	wait for background processes to complete

- Type ***help command_name*** for more info on the above commands.

Exercise:

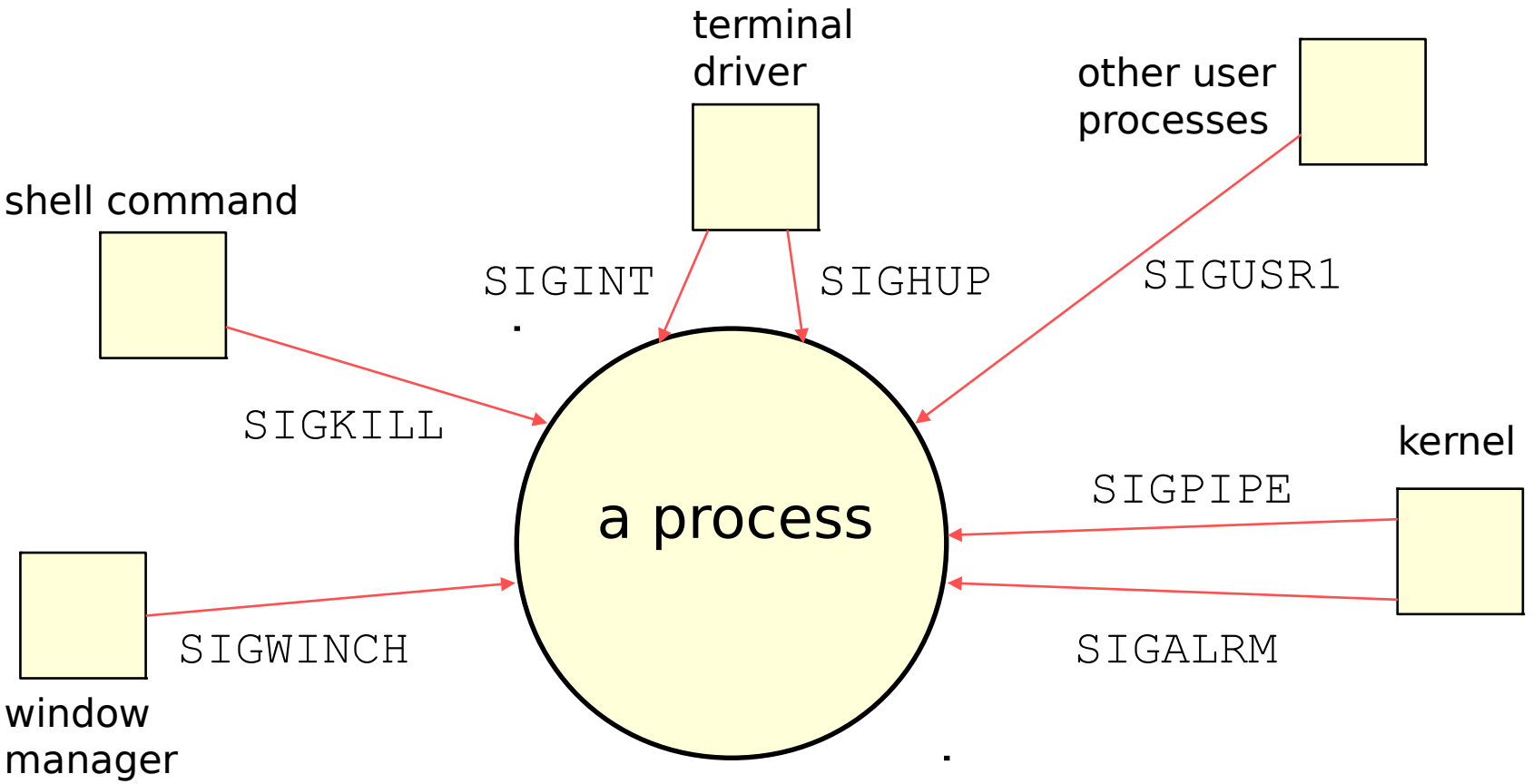
```
$ ping localhost
# ^c (press Ctrl-c)
$ jobs
```

```
$ ping localhost
# ^z (press Ctrl-z)
$ jobs
$ kill %1
```

Process Communication Using Signals

- A signal is a notification to a process that some event has occurred.
- Signals occur asynchronously, meaning that a process does not know in advance when a signal will arrive.
- Different types of signals are intended to notify processes of different types of events.
- Each type of signal is represented by an integer, and alternatively referred to by a name as well.
- Various conditions can generate signals. Some of them include:
  - The ‘kill’ command
  - Certain terminal characters (e.g. ^C is pressed)
  - Certain hardware conditions (e.g. the modem hangs)
  - Certain software conditions (e.g. division by zero)

Sources of Signals



Common Signal Types

Name	*Description*	*Default Action*
`SIGINT`	Interrupt character typed (`^C`)	terminate process
`SIGQUIT`	Quit character typed	create core image
`SIGKILL`	`kill -9`	terminate process
`SIGSEGV`	Invalid memory reference	create core image
`SIGPIPE`	Write on pipe but no reader	terminate process
`SIGALRM`	`alarm()` clock ‘rings’	terminate process
`SIGUSR1`	user-defined signal type	terminate process
`SIGUSR2`	user-defined signal type	terminate process

- See `man 7 signal`

The Bash Shell

What is a Shell?

- The shell is command language interpreter that executes commands read from standard input or from a file.
- There are several variants of shell. Ada uses bash as its default shell and our focus.

Example of What a Shell Does

```
$ cat *.txt | wc > txt_files_size.$USER
$ echo "Date? `date`" >> txt_files_size.$USER
```

1) command line is broken into "words" (or "tokens")
2) quotes are processed
3) redirection and pipes are set up
4) variable substitution takes place
5) command substitution takes place
6) filename substitution (globbing) is performed
7) command/program execution

Shell Initialization Files

- The bash shell has a number of startup files that are sourced. Sourcing a file causes all settings in the file to become a part of the current shell (no new subshell process is created).
- The specific initialization files sourced depend on whether the shell is a login shell, an interactive shell, or a non-interactive shell (a shell script).
  - When a user logs on, before the shell prompt appears, the system-wide initialization file ***/etc/profile*** is sourced
  - Next, if it exists, the ***.bash_profile*** file in the user's home directory is sourced (sets the user's aliases, functions, and environment vars if any)
  - If ***.bash_profile*** doesn't exist, but a ***.bash_login*** file does exist, it is sourced
  - If even the ***.bash_login*** doesn't exist, but a ***.profile*** does exist, it is sourced

Types of UNIX Commands

- A command entered at the shell prompt can be one of several types
  - **Alias:** an abbreviation or a ‘nickname’ for an existing command; user definable
  - **Built-in:** part of the code of the shell program; fast in execution. For more info on any particular built-in, type ***`help built-in_command_name`***
  - **Function:** groups of commands organized as separate routines, user definable, reside in memory, relatively fast in execution
  - **External program**
    - Interpreted script: like a DOS batch file, searched and loaded from disk, executed in a separate process
    - Compiled object code: searched and loaded from disk, executed in a separate process

The Exit Status of a Process

- After a command or program terminates, it returns an “exit status” to the parent process.
- The exit status is a number between 0 and 255.
  - By convention, exit status 0 means successful execution
  - Non-zero status means the command failed in some way
  - If command not found by shell, status is 127
  - If command dies due to a fatal signal, status is 128 + sig #
- After command execution, type ***`echo $?`*** at command line to see its exit status number.

```
$ grep user_NetID /etc/passwd
user_NetID:x:1234:100:user_NetID Jackson:/home/user_NetID:/bin/bash
$ echo $?
0
$ grep billclinton /etc/passwd
$ echo $?
1
```

Multiple Commands and Grouping

- A single command line can consist of multiple commands. Each command must be separated from the previous by a semicolon. The exit status returned is that of the last command in the chain.

```
$ ls; pwd; date
```

- Commands can also be grouped so that all of the output is either piped to another command or redirected to a file.

```
$ ( ls; pwd; date ) > outputfile
```

Conditional Execution and Backgrounding

- Two command strings can be separated by the special characters **&&** or **||** . The command on the right of either of these metacharacters will or will not be executed based on the exit status of the command on the left.

```
$ cc program1.c -o program1.exe && program1.exe
$ cc program2.c -o program2.exe >& err || mail bob@tamu.edu < err
```

- By placing an **&** at the end of a command line, the user can force the command to run in the "background". User will not have to wait for command to finish before receiving the next prompt from the shell.

```
$ program1.exe > p1_output &
[1] 1557
$ echo $!
1557
```

Command Line Shortcuts

- Command and filename completion
  - A feature of the shell that allows the user to type partial file or command names and completes the rest itself
- Command history
  - A feature that allows the user to “scroll” through previously typed commands using various key strokes
- Aliases
  - A feature that allows the user to assign a simple name even to a complex combination of commands
- Filename substitution
  - Allows user to use “wildcard” characters (and other special characters) within file names to concisely refer to multiple files with simple expressions

Command and Filename Completion

- To save typing, bash provides a mechanism that allows the user to type part of a command name or file name, press the tab key, and the rest of the word will be completed for the user.

```
$ ls
file1 file2 foo foobarckle fumble
$ ls fu[tab]                    (expands fu to fumble)
$ ls fx[tab]                    (terminal beeps, nothing happens)
$ ls fi[tab]                    (expands fi to file)
$ ls fi[tab][tab]               (lists all possibilities)
file1 file2

$ da[tab]                       (expands to the date command)
```

The '***history***' Command

```
$ history | tail

  994  pstree -p 27688
  995  ps
  996  ps uf
  997  ps -uf
  998  ps -l
  999  icc -o funprog -L/scratch/user/user_NetID/WhizBang/lib -lwhiz -lbang funprog.c
foo.o bar.o
 1000  echo $PATH
 1001  gedit funprog.c
 1002  history


$ !icc
icc -o funprog -L/scratch/user/user_NetID/WhizBang/lib -lwhiz -lbang funprog.o foo.o
bar.o

$ !1001
icc -o funprog -L/scratch/user/user_NetID/WhizBang/lib -lwhiz -lbang funprog.o foo.o
bar.o
```

The '*!*' Command

- The *!* (bang) can also be used for re-execution of previous commands.
- How it is used:

!!	re-execute the previous command (the most recent command)
!N	re-execute the Nth command from the history list
!string	re-execute last command starting with string
!N:s/old/new/	in previous Nth command, substitute all occurrences of old string with new string

Aliases

- An alias is a bash user-defined abbreviation for a command.
- Aliases are useful if a command has a number of options or arguments or if the syntax is difficult to remember.
- Aliases set at the command line are not inherited by subshells. They are normally set in the *.bashrc* startup file.

Aliases

- The ***alias*** built-in command lists all aliases that are currently set.

```
$ alias
alias co='compress'
alias cp='cp -i'
alias mroe='more'
alias ls='ls -F'
```

- The ***alias*** command is also used to set an alias.

```
$ alias co=compress
$ alias cp='cp -i'
$ alias m=more
$ alias mroe='more'
$ alias ls='ls -alF'
```

- The ***unalias*** command deletes an alias. The \ character can be used to temporarily turn off an alias.

```
$ unalias mroe
$ \ls
```

Filename Substitution

- Metacharacters are special characters used to represent something other than themselves.
- When evaluating the command line the shell uses metacharacters to abbreviate filenames or pathnames that match a certain set of characters.
- The process of expanding metacharacters into filenames is called filename substitution, or globbing. (This feature can be turned off with ***`set noglob`*** or ***`set -f`***)
- Metacharacters used for filename substitution:

`*`	matches 0 or more characters
`?`	matches exactly 1 character
`[abc]`	matches 1 character in the set: a, b, or c
`[!abc]`	matches 1 character not in the set: anything other than a, b, or c
`[!a-z]`	matches 1 character not in the range from a to z
`{a,ile,ax}`	matches for a character or a set of characters

Filename Substitution

```
$ ls *
abc abs1 abc122 abc123 abc2 file1 file1.bak file2 file2.bak none
nonsense nobody nothing nowhere one
$ ls *.bak
file1.bak file2.bak
$ echo a*
abc abc1 abc122 abc123 abc2
$ ls a?c?
abc1 abc2
$ ls ??
ls: ??: No such file or directory
$ echo abc???
abc122 abc123
$ echo ??
??
$ ls abc[123]
abc1 abc2
$ ls abc[1-3]
abc1 abc2
$ ls [a-z][a-z][a-z]
abc one
```

Filename Substitution

```
$ ls *
a.c b.c abc ab3 ab4 ab5 file1 file2 file3 file4 file5 foo faa fumble
$ ls f{oo,aa,umble}
foo faa fumble
$ ls a{.c,c,b[3-5]}
a.c ab3 ab4 ab5
$ mkdir /home/user_NetID/mycode/{old,new,dist,bugs}
$ echo fo{o, um}*
fo{o, um}*
```

- To use a metacharacter as a literal character, use the backslash to prevent the metacharacter from being interpreted.

```
$ ls
abc file1 youx
$ echo How are you?
How are youx
$ echo How are you\?
How are you?
$ echo When does this line \
>ever end\?
When does this line ever end?
```

Filename Substitution

- The tilde character (~) by itself evaluates to the full pathname of the user's home directory.
- When prepended to a username, the tilde expands to the full pathname of that user's home directory.
- When the plus sign follows the tilde, the value of the present working directory is produced.
- When the minus sign follows, the value of the previous working directory is produced.

```
$ echo ~
/home/user_NetID
$ echo ~fdang
/home/fdang
$ echo ~+
/home/user_NetID
$ echo ~-
/home/user_NetID/mycode
```

Command Substitution

- Used when:
  - assigning the output of a command to a variable
  - substituting the output of a command within a string
- Two ways to perform command substitution
  - placing the command within a set of backquotes
  - placing the command within a set of parenthesis preceded by a $ sign
- Bash performs the substitution by executing the command and returning the standard output of the command, with any trailing newlines deleted.

Command Substitution

```
$ echo "The hour is `date +%H`"
The hour is 12
$ set `date`
$ echo $*
Mon Sep 13 12:25:46 CDT 2004
$ echo $2 $6
Sep 2004
$ d=$(date)
$ echo $d
Mon Sep 13 12:27:40 CDT 2004
$ lines=$(cat file1)
$ echo The time is $(date +%H)
The time is 12
$ machine=$(uname –n)
$ echo $machine
login1.tamu.edu
$ pwd
/home/user_NetID
$ dirname="$(basename $(pwd))"
$ echo $dirname
user_NetID
```

File Descriptors & I/O Streams

- Every process needs to communicate with a number of outside entities in order to do useful work
  - It may require input to process. This input may come from the keyboard, from a file stored on disk, from a joystick, etc.
  - It may produce output resulting from its work. This data will need to be sent to the screen, written to a file, sent to the printer, etc.
- In unix, anything that can be read from or written to can be treated like a file (terminal display, file, keyboard, printer, memory, network connection, etc.)
- Each process references such “files” using small unsigned integers (starting from 0) stored in its file descriptor table.
- These integers, called file descriptors, are essentially pointers to sources of input or destinations for output.

I/O Redirection

- When an interactive shell (which of course runs as a process) starts up, it inherits 3 I/O streams by default from the login program:
  - *stdin* normally comes from the keyboard (fd 0)
  - *stdout* normally goes to the screen (fd 1)
  - *stderr* normally goes to the screen (fd 2)
- However, there are times when the user wants to read input from a file (or other source) or send output to a file (or other destination). This can be accomplished using I/O redirection and involves manipulation of file descriptors.

Redirection Operators

< redirects input

> redirects output

>> appends output

<< input from *here document*

2> redirects error

&> redirects output and error

>& redirects output and error

2>&1 redirects error to where output is going

1>&2 redirects output to where error is going

Redirection Operators

```
$ tr '[A-Z]' '[a-z]' < myfile

$ ls > lsfile
$ cat lsfile
dir1
dir2
file1
file2
file3

$ date >> lsfile
$ cat lsfile
dir1
dir2
file1
file2
file3
Mon Sep 13 13:29:40 CDT 2004

$ cc prog.c 2> errfile          (save any compilation errors in file errfile)

$ find . -name \*.c -print > foundit 2> /dev/null   (redirect output to foundit and throw away errors)
```

Pipes

- A pipe takes the output of the command to the left of the pipe symbol (|) and sends it to the input of the command listed to its right.
- A ‘pipeline’ can consist of more than one pipe.

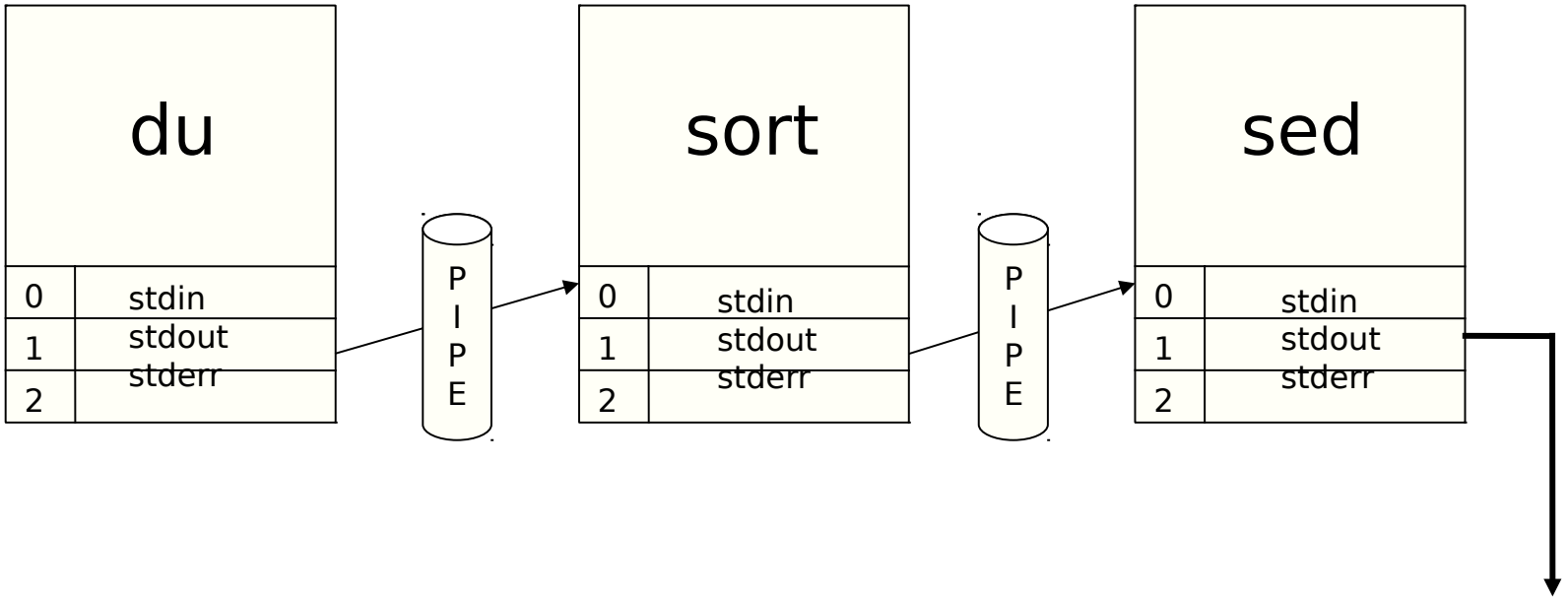
```
$ who > tmp
$ wc -l tmp
38 tmp
$ rm tmp
```

(using a pipe saves disk space and time)

```
$ who | wc -l
38
$ du . | sort -n | sed -n '$p'
84480   .
```

Pipes

```
$ du . | sort -n | sed -n '$p'
84480   .
```



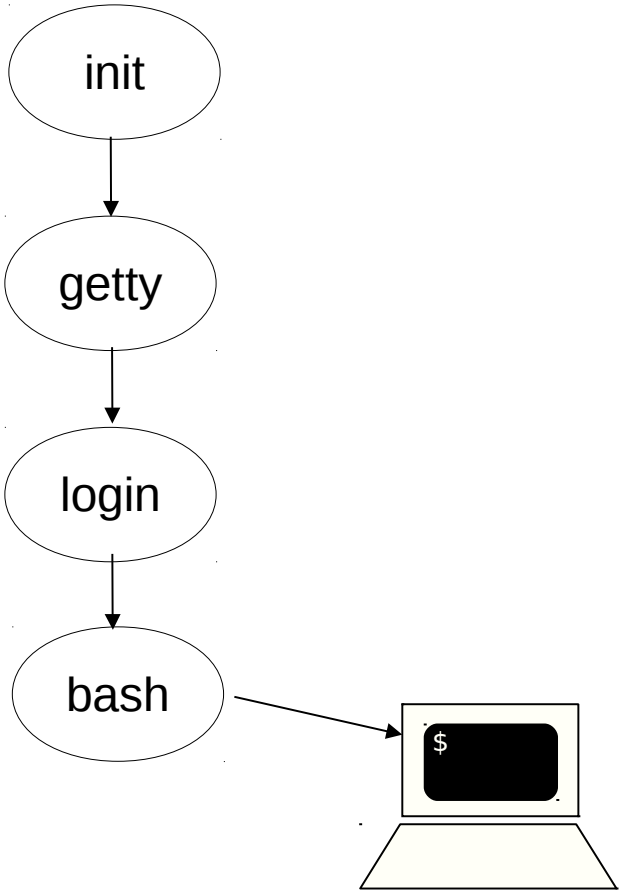
Transfer data between Unix hosts with ***scp***

```
$scp [[user@]host1:]filename1 [[user@]host2:]filena2
```

```
$ scp myfile1 user@ada.tamu.edu:
$ scp myfile1 \ user@ada.tamu.edu:dir1/remote_myfile1
$ scp user@ada.tamu.edu:myfile2 .
$ scp user@ada.tamu.edu:myfile2 \ local_myfile2
$ scp -r user@ada.tamu.edu:dir3 local_dir/   (copy recursively)
```

Recap: Bash Shell Initialization

The Startup of the Bash Shell

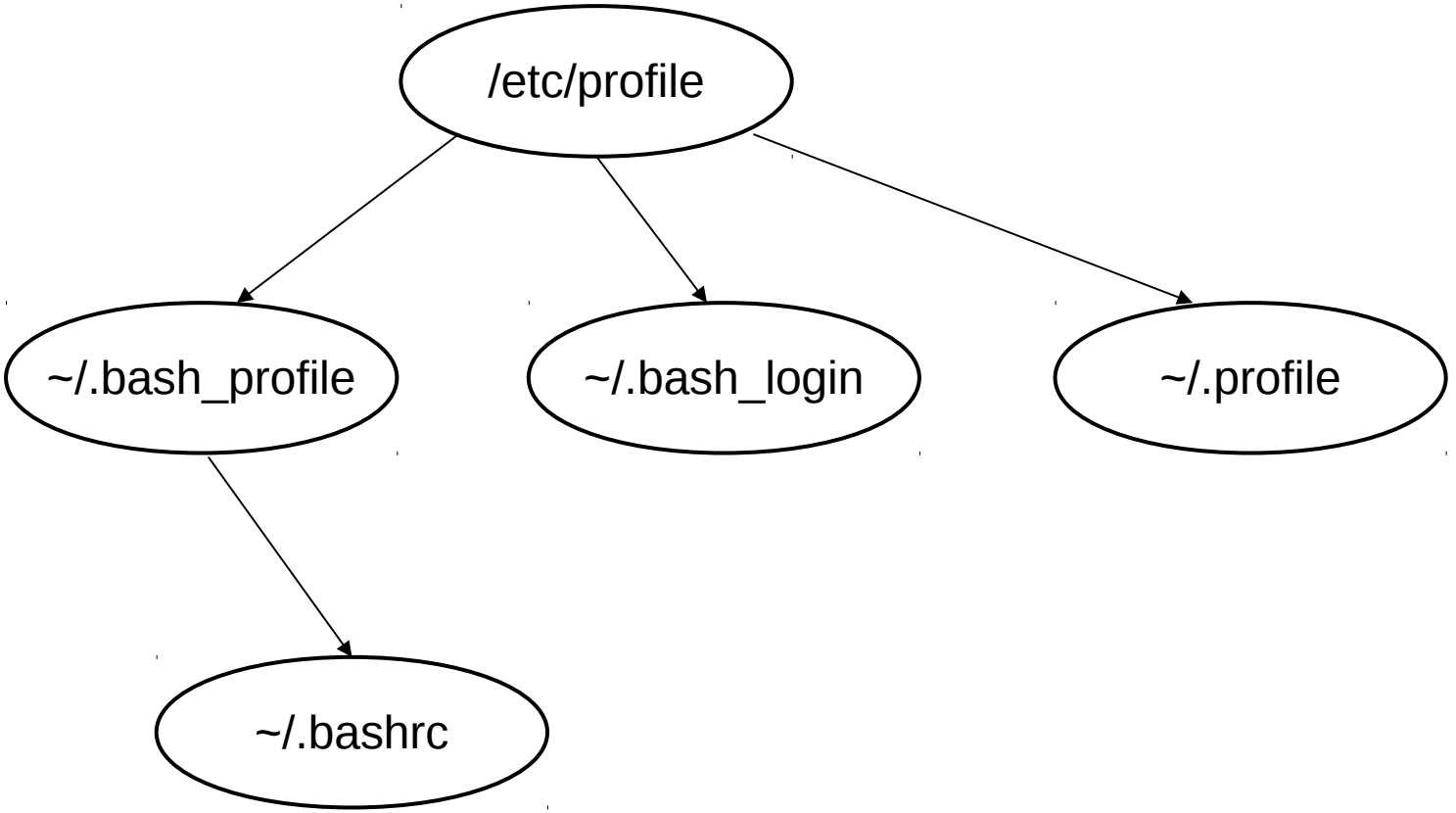


- When system boots, 1st process to run is called *init* (PID 1).
- It spawns *getty*, which opens up terminal ports and puts a login prompt on the screen.
- */bin/login* is then executed. It prompts for a password, encrypts and verifies the password, sets up the initial environment, and starts the login shell.
- *bash*, the login shell in this case, then looks for and executes certain startup files that further configure the user's environment.

Shell Initialization Files

- The bash shell has a number of startup files that are sourced. Sourcing a file causes all settings in the file to become a part of the current shell (no new subshell process is created).
- The specific initialization files sourced depend on whether the shell is a login shell, an interactive shell, or a non-interactive shell (a shell script).
  - When a user logs on, before the shell prompt appears, the system-wide initialization file ***/etc/profile*** is sourced
  - Next, if it exists, the ***.bash_profile*** file in the user's home directory is sourced (sets the user's aliases, functions, and environment vars if any)
  - If ***.bash_profile*** doesn't exist, but a ***.bash_login*** file does exist, it is sourced
  - If even the ***.bash_login*** doesn't exist, but a ***.profile*** does exist, it is sourced

Shell Initialization Files



The ***.bashrc*** file, if it exists, is sourced every time an interactive bash shell or script is started.

Queue Limits

Queue	Min/Default/Max Cpus	Default/Max Walltime	Compute Node Types	Pre-Queue Limits	Aggregate Limits Across Queues	Per-User Limits Across Queues	Notes
sn_short	1 / 1 / 20	10 min / 1 hr	64 GB nodes (811) 256 GB nodes (26)		Maximum of **6000** cores for all running jobs in the single-node (sn_*) queues.	Maximum of **1000 cores and 50 jobs per user** for all running jobs in the single node (sn_*) queues.	For jobs needing **more than one compute node**.
sn_regular		1 hr / 1 day					
sn_long		24 hr / 4 day					
sn_xlong		4 days / 7 days					
mn_short	2 / 2 / 200	10 min / 1hr		Maximum of **2000** cores for all running jobs in this queue.	Maximum of **12000** cores for all running jobs in the multi-node (mn_*) queues.	Maximum of **3000 cores and 150 jobs per user** for all running jobs in the multi-node (mn_*) queues.	For jobs needing **more than one compute node**.
mn_small	2 / 2 / 120	1 hr / 7 days		Maximum of **6000** cores for all running jobs in this queue.			
mn_medium	121 / 121 / 600	1 hr / 7 days		Maximum of **6000** cores for all running jobs in this queue.			
mn_large	600 / 601 / 2000	1 hr / 5 days		Maximum of **5000** cores for all running jobs in this queue.			
xlarge	1 / 1 / 280	1 hr / 10 days	1 TB nodes (11) 2 TB nodes (4)				For jobs needing **more than 256GB of memory per compute node**.
vnc	1 / 1 / 20	1 hr / 6 hr	GPU nodes (30)				For remote visualization jobs.
special	None	1 hr / 7 days	64 GB nodes (811) 256 GB nodes (26)				Requires permission to access this queue.

Job Files (Serial Example)

```
#BSUB -J myjob1           # sets the job name to myjob1.
#BSUB -L /bin/bash        # uses the bash shell to initialize the job's execution environment
#BSUB -W 12:30            # sets to 12.5 hours the job's runtime wall-clock limit.
#BSUB -n 1                # assigns 1 core for execution.
#BSUB -o stdout1.%J       # directs the job's standard output to stdout1.jobid
#BSUB -M 500              # specifies a memory limit of 500 MB per core/process
#BSUB -R “rusage[mem=500]”   # requests 500MB of memory per core/process from LSF
#
# <--- at this point the current working directory is the one you submitted the job from.
#
module load intel/2015B   # loads the Intel software tool chain to provide, among other things,
#                         needed runtime libraries for the execution of prog.exe below.
#                         (assumes prog.exe was compiled using Intel compilers.)
#
prog.exe < input1 >& data_out1   # both input1 and data_out1 reside in the job submission dir
##
```

https://hprc.tamu.edu/wiki/index.php/Ada:Batch#Job_files

Job Resource Examples

#BSUB -n 10 -R “span[ptile=2]”

#BSUB -R “rusage[mem=500]” -M 500 ...

Requests 10 job slots (2 per node). The job will span 5 nodes. The job can use up to 1000 MB per node.

#BSUB -n 80 -R “span[ptile=20]”

#BSUB -R “rusage[mem=2500]” -M 2500

Request 4 whole nodes (80/20), not including the xlarge memory nodes. The job can use up to 50GB per node.

Job Parameters Example

```
#BSUB -L /bin/bash                    # use the bash login shell
#BSUB -J stacks_S2                    # job name
#BSUB -n 20                           # assigns 20 cores for execution
#BSUB -R "span[ptile=20]"             # assigns 20 cores per node
#BSUB -R "select[mem256gb]"           # nxt, mem256gb, mem1tb, mem2tb, ...
#BSUB -R "rusage[mem=12400]"          # reserves 12400MB memory per core
#BSUB -M 12400                        # sets to 12400MB process limit
#BSUB -W 1:00                         # sets to 1 hour the job's limit
#BSUB -o stdout.%J                    # job standard output to stdout.jobid
#BSUB -e stderr.%J                    # job standard error to stderr.jobid
```

Optional; only needed if you want to use higher memory nodes, otherwise defaults to the 64GB nodes

Job Parameters Example
For Job Scripts on xlarge queue

```
#BSUB -L /bin/bash                    # use the bash login shell
#BSUB -J stacks_S2                    # job name
#BSUB -n 40                           # assigns 40 cores for execution
#BSUB -R "span[ptile=40]"             # assigns 40 cores per node
#BSUB -q xlarge                       # required if using mem1tb or mem2tb
#BSUB -R "select[mem1tb]"             # nxt, mem256gb, mem1tb, mem2tb, ...
#BSUB -R "rusage[mem=25000]"          # reserves 25GB memory per core
#BSUB -M 25000                        # sets to 25GB process limit
#BSUB -W 48:00                        # sets to 48 hours the job's limit
#BSUB -o stdout.%J                    # job standard output to stdout.jobid
#BSUB -e stderr.%J                    # job standard error to stderr.jobid
```

These two parameters must be specified together
if you are using the 1TB or 2TB computational nodes

Job Files (Concurrent Serial Runs)

#BSUB -J myjob2 # sets the *job name* to myjob2.

#BSUB -L /bin/bash # uses the bash *login shell* to initialize the job's execution environment.

#BSUB -W 12:30 # sets to 12.5 hours the job's runtime *wall-clock limit*.

#BSUB -n 3 # assigns *3 cores* for execution.

#BSUB -R "span[ptile=3]" # assigns *3 cores per node*.

#BSUB -R "rusage[mem=5000]" # *reserves 5000MB per process*/CPU for the job (i.e., 15,000 MB for job/node)

#BSUB -M 5000 # sets to 5,000MB (~5GB) the per process *enforceable memory limit*.

#BSUB -o stdout2.%J # directs the job's *standard output* to stdout2.jobid

#BSUB -P project1 # charges the consumed *service units* (SUs) to project1.

#BSUB -u e-mail_address # sends *email* to the specified address

#BSUB -B -N # send emails on *job start* (-B) and *end* (-N)

cd $SCRATCH/myjob2 # makes $SCRATCH/myjob2 the job's current *working directory*

module load intel/2015B # loads the **Intel** software tool chain to provide, among other things,

The next 3 lines concurrently execute 3 instances of the same program, prog.exe, with standard input and output data streams assigned to different files in each case.

(prog.exe < input1 >& data_out1) &

(prog.exe < input2 >& data_out2) &

(prog.exe < input3 >& data_out3)

wait

https://hprc.tamu.edu/wiki/index.php/Ada:Batch#Job_files

Pop Quiz

```
#BSUB -L /bin/bash
#BSUB -J stacks_S2
#BSUB -n 10
#BSUB -R "span[ptile=10]"
#BSUB -R "rusage[mem=2000]"
#BSUB -M 2000
#BSUB -W 36:00
#BSUB -o stdout.%J
#BSUB -e stderr.%J
```

How much total memory is requested for this job?

What is the maximum time this job is allowed to run?

Pop Quiz

```
#BSUB -L /bin/bash
#BSUB -J stacks_S2
#BSUB -n 80
#BSUB -R "span[ptile=80]"
#BSUB -R "select[mem1tb]"
#BSUB -R "rusage[mem=25000]"
#BSUB -M 25000
#BSUB -W 48:00
#BSUB -o stdout.%J
#BSUB -e stderr.%J
```

Find two parameters that are either missing or not configured correctly

Exercises

- Copy sample job scripts under /scratch/training/Digital_Biology
  - example01.env_variables.job
  - example02.echo_numbers.job (time limit)
  - example03.python_memory.job (memory limit)
  - example04.r_example.job
  - example05.bad_core_ptile.job (no resource match)
- Modify job scripts to trigger/fix errors.

Remote Visualization Jobs

- Use to run programs with graphical interfaces on Ada and display them on your computer:
- Can leverage GPU nodes for better graphics performance
- Better than X11 forwarding (especially when using VPN)

Command	Description
vncjob.submit [-h] [-g MxN] [-t type]	Submit a VNC job. Type 'vncjob.submit -h' for help
vncjob.kill JOBID	Kill a VNC job whose id is JOBID
vncjob.list	List all your VNC jobs currently in the batch system

https://hprc.tamu.edu/wiki/index.php/Ada:Remote_Visualization

Remote Visualization Job Example

(1) Log into Ada

```
Your current disk quotas are:
Disk       Disk Usage      Limit    File Usage      Limit
/home             33M        10G           676      10000
/scratch        4.533G         1T         13749      50000
/tiered              0        10T             1      50000
Type 'showquota' to view these quotas again.
[ netid@ada2 ~]$
```

(2) Submit VNC Job using vncjob.submit (optional parameters available)

```
Type 'showquota' to view these quotas again.
[ netid@ada2 ~]$ vncjob.submit
Your vnc job has been submitted.

Output file for VNC job 1551326 will be /home/      /vncjob.1551326.

View the output with the follwoing command when your job starts running

   cat /home/ netid /vncjob.1551326

For more information about remote visualization on ada, please visit

    https://sc.tamu.edu/wiki/index.php/Ada:Remote-Viz
[ netid@ada2 ~]$
```

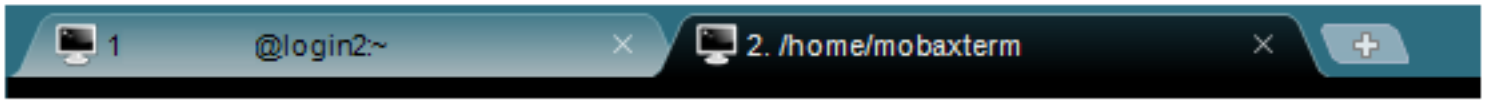
Remote Visualization Job Example

(3) Use cat to see the output file -- Note job properties

```
[ netid@ada2 ~]$ cat /home/netid /vncjob.1551326
Using settings in ~/.vnc/xstartup.turbovnc to start /opt/TurboVNC/bin/vncserver
VNC batch job id is 1551326
VNC server arguments will be '-geometry 1024x768'
VNC server started with display gpu64-3001:11

VirtualGL Client 64-bit v2.4 (Build 20150126)
Listening for unencrypted connections on port 4242
4242

WARNING: You have started an interactive/VNC job.  Your job will continue
         to run until the VNC server is stopped (up to 6 hours).

To access from Mac/Linux, run from your desktop:

    vncviewer -via  netid@ada.tamu.edu gpu64-3001:11

To access from Windows:

    1) Setup a tunnel from your machine to gpu64-3001:5911

        1.1) If you use MobaXterm, run the following command in the MobaXterm terminal:

             ssh -f -N -L 10000:gpu64-3001:5911  netid@ada.tamu.edu

        1.2) If you use Putty to set up the tunnel, click SSH' and then click 'Tunnels'.
             Fill in 'Source port' with '10000' and 'Destination' with 'gpu64-3001:5911'

    2) Start vncviewer on your machine

Otherwise to access from Windows, either see the documentation that came
with your VNC viewer, or open an X11 enabled login to ada.tamu.edu and
then run:

    vncviewer gpu64-3001:11

When running graphical program in this VNC job, remember to start them using vglrun:

    vglrun application

To stop the VNC job:

    vncjob.kill 1551326

[ netid@ada2 ~]$
```

Remote Visualization Job Example

(4) Start new tab/terminal pointed to local machine




```
[2015-07-08 14:32.50]  ~
[            Laptop] ➤
```

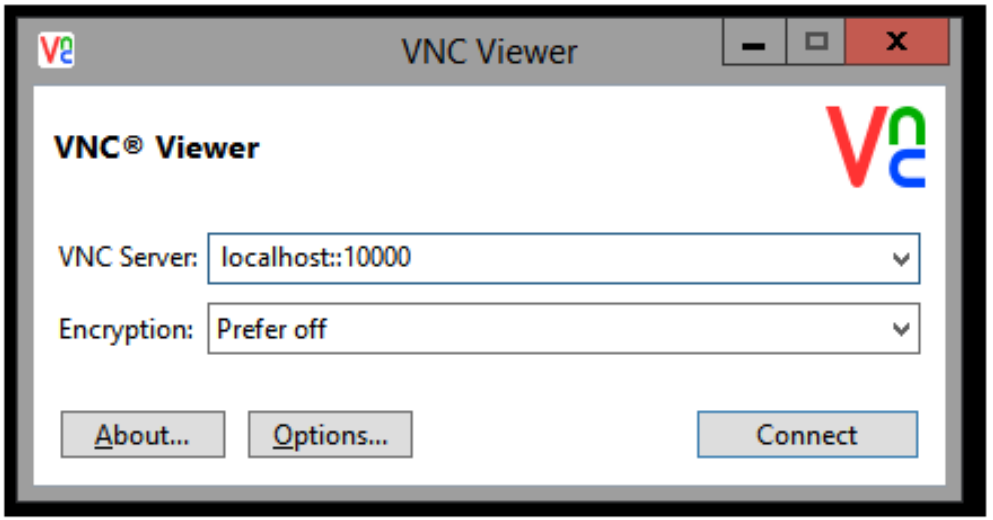
(5) Use command from (3) to create tunnel -- Local port 10000 must be free

```
[2015-07-08 14:37.55]  ~
[            Laptop] ➤ ssh -f -N -L 10000:gpu64-3001:5911   netid@ada.tamu.edu
NOTICE: This computer system and data herein are available only for
authorized purposes by authorized users. Use for any other purpose
may result in administrative/disciplinary actions or criminal prosecution
against the user. Usage may be subject to security testing and monitoring.
Applicable privacy laws establish the expectations of privacy.

 netid@ada.tamu.edu's password:
```

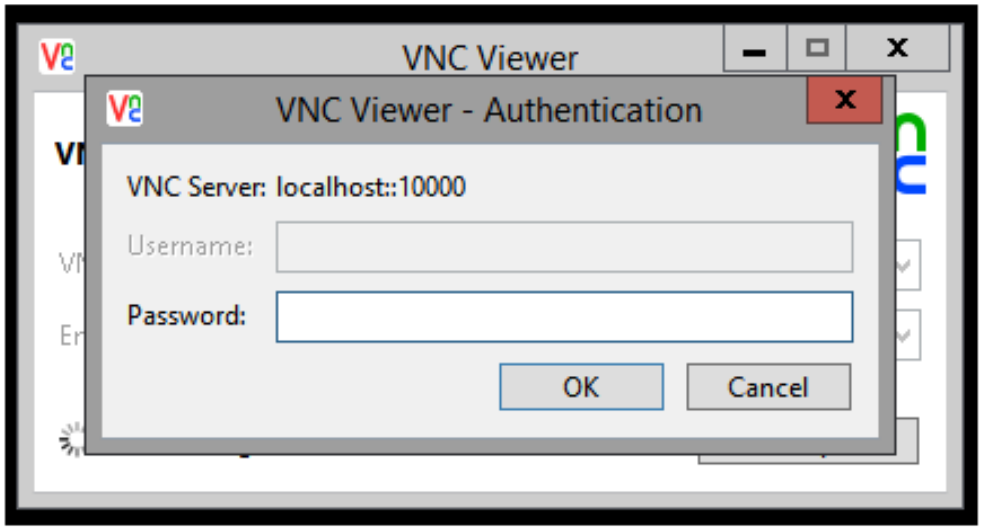
```
[2015-07-08 14:38.15]  ~
[            Laptop] ➤
```

Remote Visualization Job Example

(6) Open VNC Viewer and enter connection information



(7) Enter your *VNC* password



Remote Visualization Job Example

(8) VNC window opens -- load modules -- use vglrun to launch GUI