



# Introduction to **LINUX/SHELL** in SHARCNET

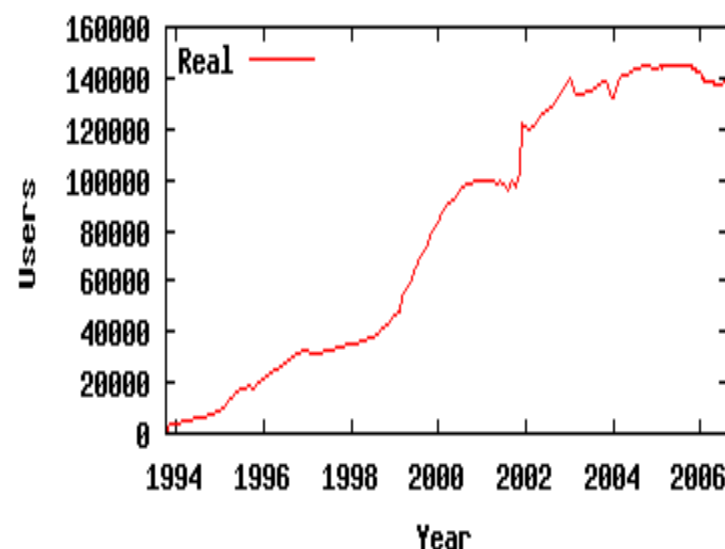
Isaac Ye, High Performance Technical Consultant  
SHARCNET, York University  
[isaac@sharcnet.ca](mailto:isaac@sharcnet.ca)

# Outlines

- **Intro to WHAT/WHY/WHICH LINUX**
- **Understanding CLUSTER environment**
- **LINUX basics**
  - **Login (SSH)**
  - **Filesystem**
  - **Basic command**
- **SHELL Intro**
  - **Basic command-line operation**
  - **Text editor (nano/vi)**
  - **Pipes**

# What is Linux?

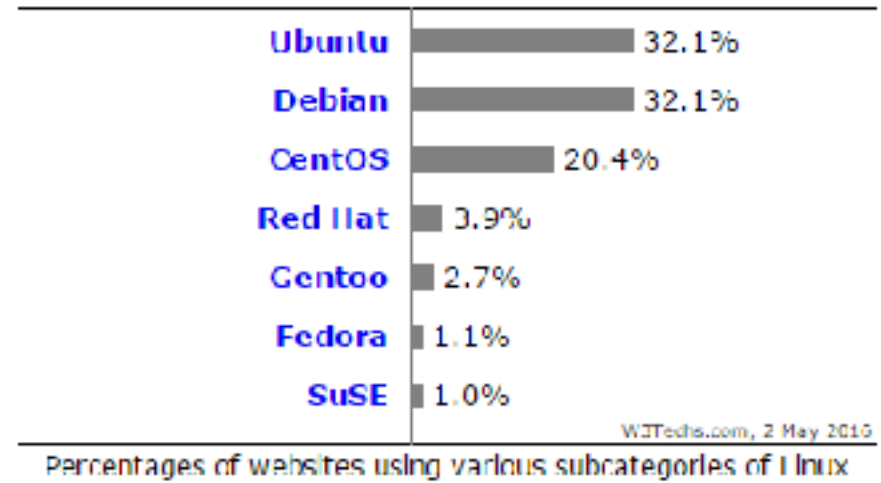
- History
  - A famous professor Andrew Tanenbaum developed **Minix**, a simplified version of UNIX that runs on PC
  - In Sept 1991, **Linus Torvalds** developed the preliminary kernel of Linux, known as Linux version 0.0.1
  - Recent (2007) estimates about 30M users in the world.
  - 95% of Top500 supercomputers running on Linux



# Why Linux?

- A fully-networked 64-Bit Unix-like OS
- Excellent system stability
- Unix tools and compilers
- Strong network tools and support
- Multi-user, Multitasking, Multiprocessor
- Has the network-based X Windows GUI
- Runs on multiple platforms(hardware)
- Plentiful software
- Includes the source code and documents
- FREE !!!

# Which Linux



- Red Hat Linux : One of the original Linux distribution.
- Debian GNU/Linux : A free software distribution.
- Ubuntu Linux: an immensely popular Debian-based distribution.
- CentOS: an Enterprise-class Linux Distribution derived from sources freely provided to the public. (SHARCNET uses)
- SuSE Linux : primarily available for pay because it contains many commercial programs.

# CLUSTER Environment - GRAHAM



# GRAHAM (I)



Hardware specification	
Processors	32136 CPUs and 320 GPUs
Interconnect	100Gb/s Mellanox FDR InfiniBand Interconnect 56Gb/s Mellanox EDR InfiniBand Interconnect
CPU Nodes	800 nodes(128GB base) : 32 cores/node
	56 nodes(256GB large): 32 cores/node
	24 nodes(512GB bigmen500): 32 cores/ node
	3 nodes(3000GB bigmen3000: 56 cores/ node
GPU Nodes	160 nodes: 24 cores/node 2 NVIDIA P100 Pascal GPUs/node



# GRAHAM (II)

- Named in honour of Prof. Wes Graham, the first director of U. Waterloo's computing centre.
- >1K nodes and 33K CPU cores
- 2.6 petaFLOPs of peak theoretical computational performance (< Top 50 in Top500 supercomputer list)
- Total 149 TB memory
- Liquid Cooling system using rear-door heat exchangers
- Big data ready – 5 petabyte parallel storage system
- Cloud computing using OpenStack

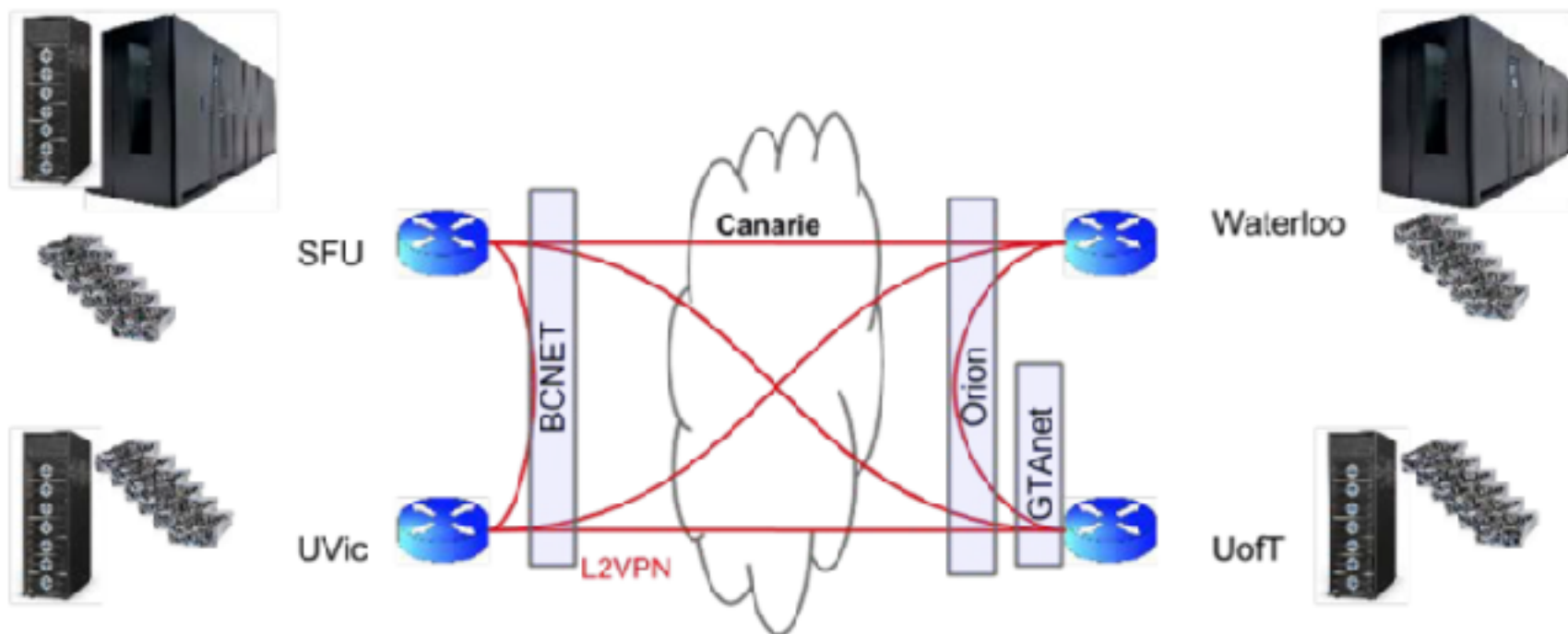




# GRAHAM- Storage

Home	Scratch	Project
<ul style="list-style-type: none"> <li>• Standard home directory.</li> <li>• Small, standard quota.</li> <li>• Not allocated via RAS or RAC. Larger requests go to Project space.</li> </ul>	<ul style="list-style-type: none"> <li>• For active or temporary (/scratch) storage.</li> <li>• Available to all nodes.</li> <li>• Not allocated.</li> <li>• Inactive data will be purged.</li> <li>• Huawei OceanStor storage system with approximately 3.6PB usable capacity and aggregate performance of approximately 30GB/s.</li> </ul>	<ul style="list-style-type: none"> <li>• Part of the National Data Cyberinfrastructure.</li> <li>• Allocated via RAS or RAC.</li> <li>• Available to all nodes.</li> <li>• Not designed for parallel I/O workloads. Use Scratch space instead.</li> </ul>

# National Data Cyberinfrastructure



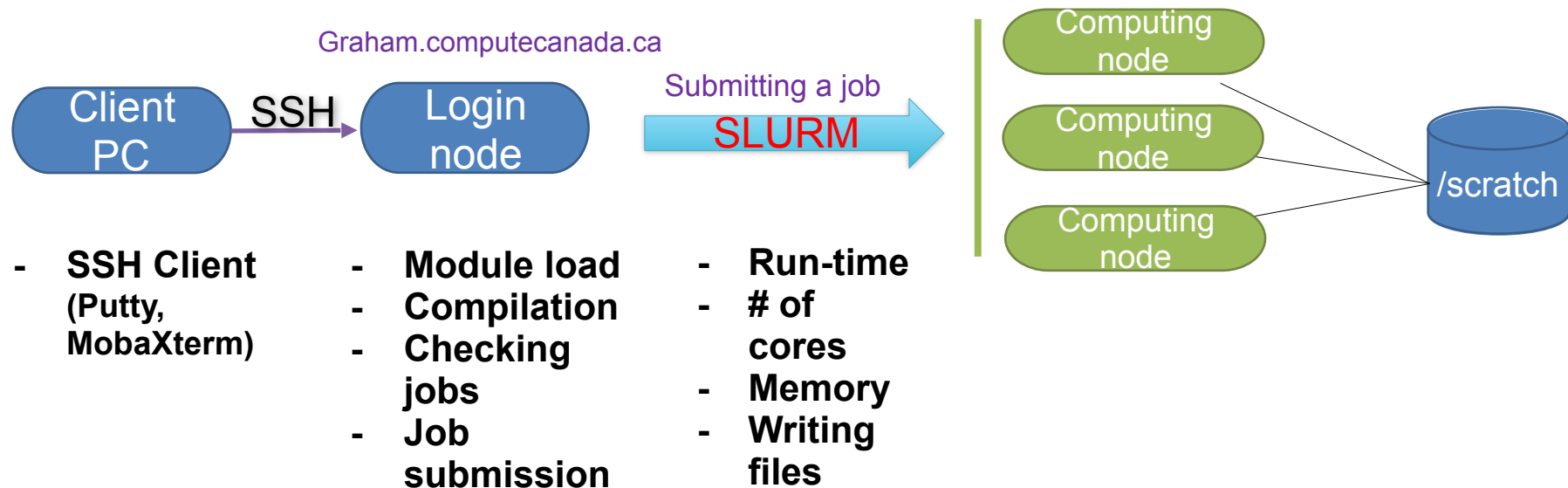
# Filesystem Quotas and Policies

Filesystem	Quotas	Backup?	Purged?	Default?	Mounted on CC
Home	50 GB, 500 K files	Yes	No	Yes	Yes
Scratch	20 TB, 1000K files 100 TB, 10M files(G)	No	Yes	Yes	Yes
Project	500K files 10 TB, 5M files(G)	Yes	No	Yes (certain amount)	Yes
Nearline (Tape)	5 TB per group	Yes	No	No	No

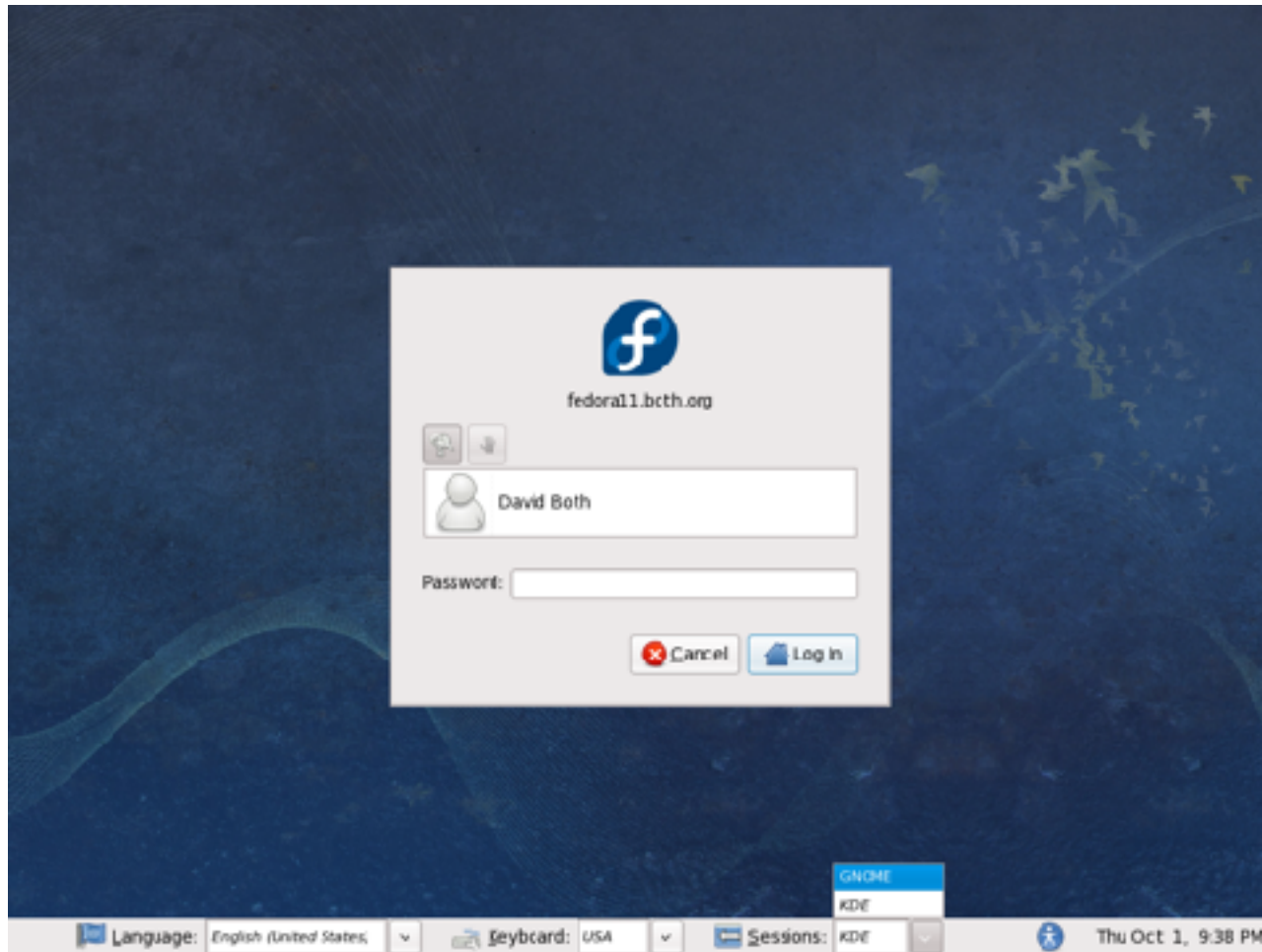
# Software environment

- This applies to new national CC systems, in particular to cedar and graham
- Operating system: LINUX - CentOS 7
- Languages for development: C/C++, Fortran, Python, Java, Matlab, all available in different versions and flavours

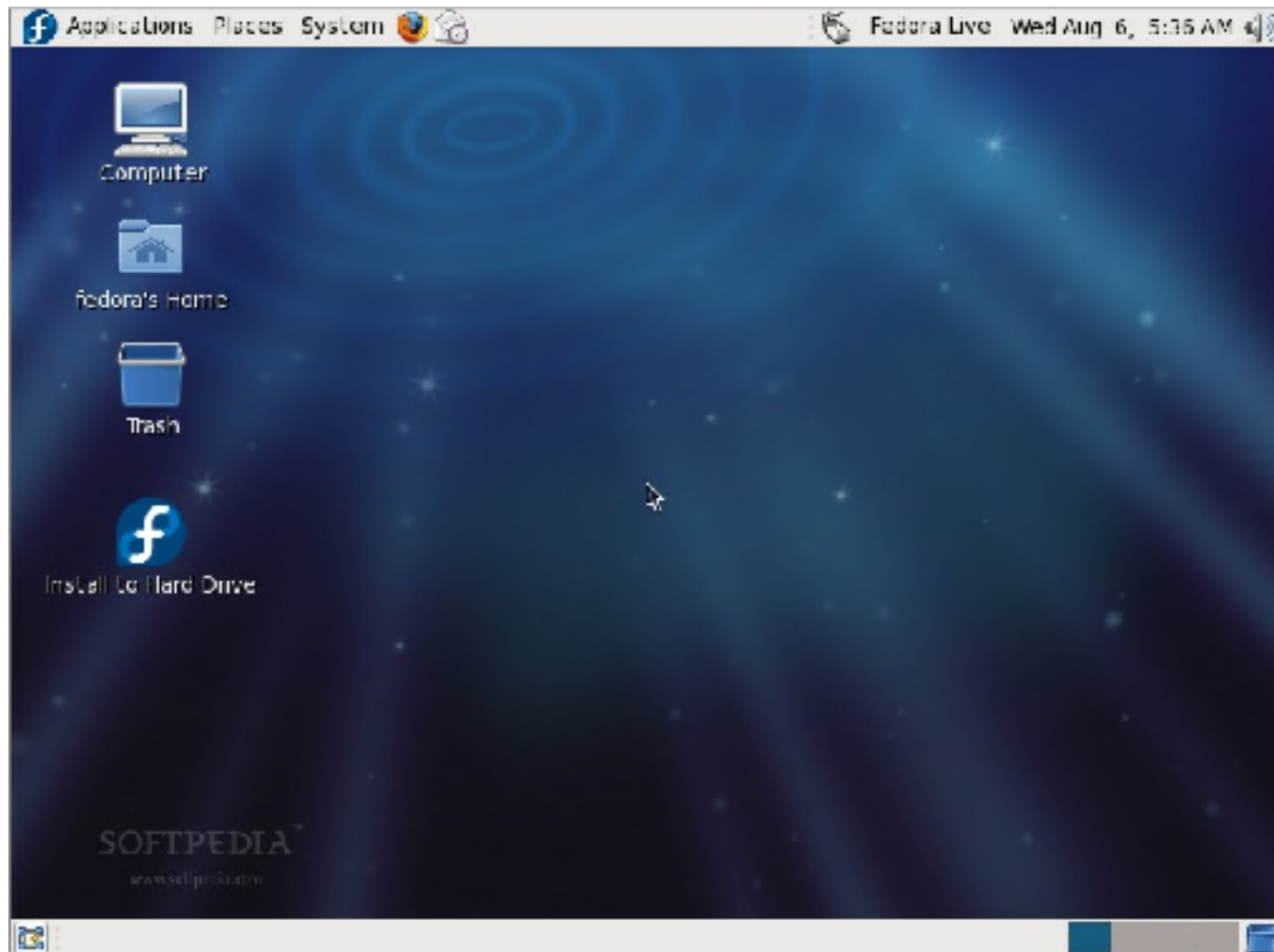
# JOB, SCHEDULER, RESOURCE



# Logging In/Out in Desktop



# Desktop snapshot





# Logging In/Out to a Server

- Connect to the server (SSH only in SHARCNET)

```
[isaac@cfdp8 isaac]$ ssh isaac@saw.sharcnet.ca
isaac@saw.sharcnet.ca's password:
Last login: Tue May 25 11:36:11 2010 from bas9-toronto12-1128700169.dsl.bell.ca
```

← Last login info

```
Welcome to Saw, a SHARCNET cluster.
Please see the following URL for status of this and other clusters:
https://www.sharcnet.ca/my/systems
```

← Welcome message

```
****
```

```
ALL Sharcnet users must now also have a Compute Canada account. Please
visit http://ccdb.sharcnet.ca for instructions.
```

```
****
```

```
[isaac@saw377 ~]$
```

- Exit from the server (Don't forget !)

← Command prompt

```
[isaac@saw377 ~]$ exit
logout
```

# The Command Prompt

- Commands are the way to “do things” in Unix
- A command consists of a command name and options called “flags”
- Commands are typed at the *command prompt*
- In Unix, *everything* (including commands) is case-sensitive

**[prompt]** \$ <command> <flags> <args>

[isaac@saw377 ~]\$ ls -l -a my\_project

↑  
Command Prompt

↑  
Command

↑  
↑  
↑  
(Optional) flags

↑  
(Optional) arguments

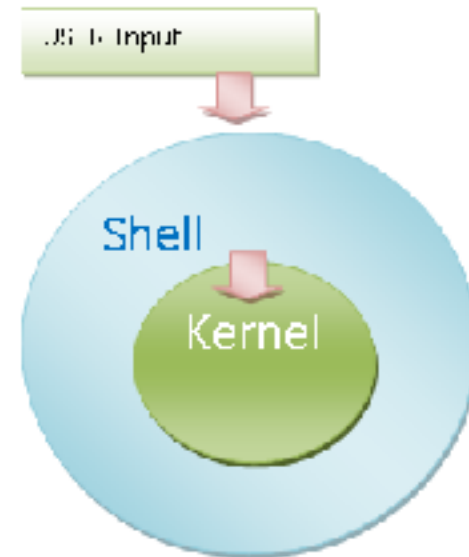
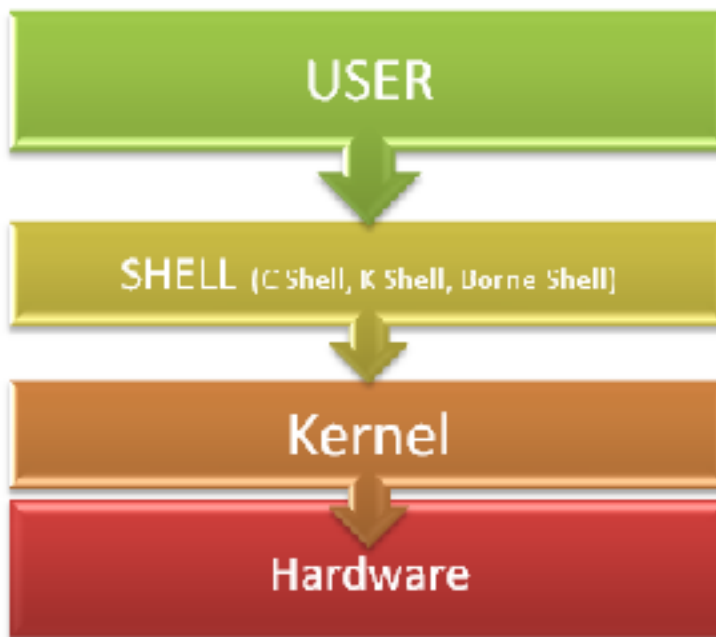
**Note:** In Unix, you’re expected to know what you’re doing. Many commands will print a message only if something went wrong.

# Two Basic Commands for Help

- The most useful commands you'll ever learn:
  - `man` (short for “*manual*”)
  - `info`
- They help you find information about other commands
  - `man <cmd>` or `info <cmd>` retrieves detailed information about `<cmd>`
  - `man -k <keyword>` searches the man page summaries (faster, and will probably give better results)
  - `man -K <keyword>` searches the full text of the man pages
- *command --help*
  - `[isaac@saw377 ~]$ ls --help`
  - Usage: `ls [OPTION]... [FILE]...`
  - List information about the FILES (the current directory by default).
  - Sort entries alphabetically if none of `-cftuvSUX` nor `--sort...`

# What is Shell?

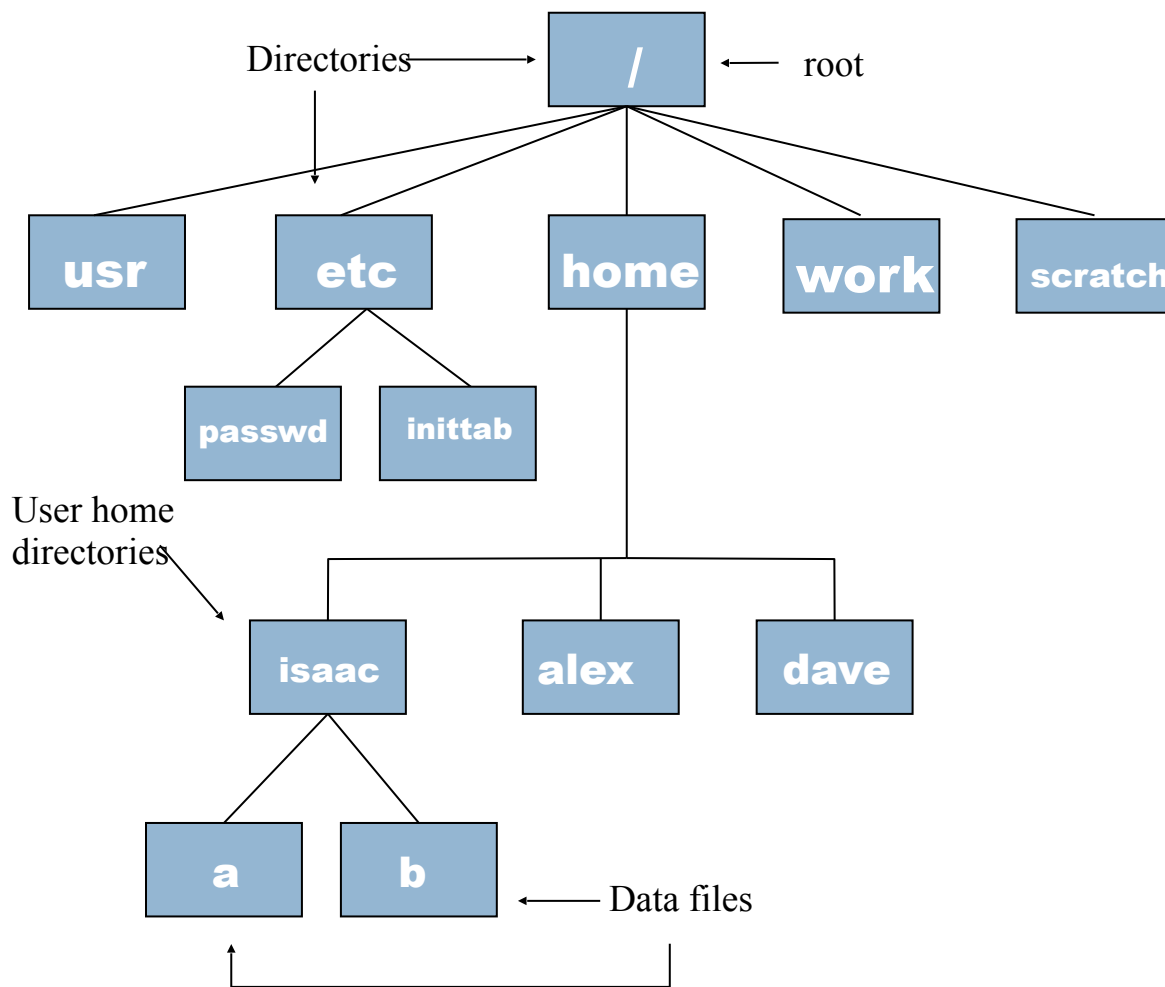
- Shell is the interface between end user and the system



```
[isaac@saw377 ~]$ echo $SHELL  
/bin/bash
```

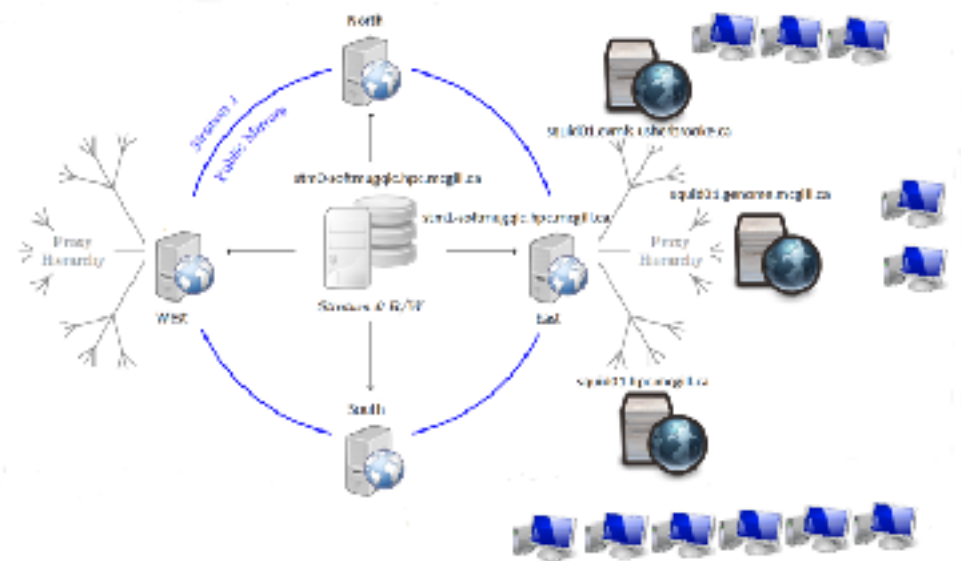
# Linux File System Basics

- Linux files are stored in a single rooted, hierarchical file system
- Data files are stored in directories (folders)
- Directories may be nested as deep as needed



# What's new in Filesystem?

- The CernVM File System provides a scalable, reliable and low-maintenance software distribution service
- All installations are done centrally at one location and automatically distributed to all remote resource centers. Basically an install once, run anywhere ability is provided.



# What's new in Module?

- **LMOD** tool developed at TACC replaces 'Environment modules'(flat structure) in most legacy servers
- LMOD is a Lua based module system that easily handles the MODULEPATH Hierarchical problem
- Similar to the module in legacy system
- Supports flat layout of modules and software hierarchy
- A "modulefile" contains the information needed to make an application or library available in the user's login session. Typically a module file contains instructions that modify or initialize environment variables such as PATH and LD\_LIBRARY\_PATH in order to use different installed programs.



# The Command Prompt

- Commands are the way to “do things” in UNIX
- A command consists of a command name and options called “flags”
- Commands are typed at the *command prompt*
- In Unix, *everything* (including commands) is case-sensitive

**[prompt]** \$ <command> <flags> <args>

[isaac@saw377 ~]\$ ls -l -a my\_project

↑  
Command Prompt

↑  
Command

↑  
↑  
↑  
(Optional) flags

↑  
(Optional) arguments

**Note:** In UNIX, you’re expected to know what you’re doing. Many commands will print a message only if something went wrong.

# Two Basic Commands for Help

- The most useful commands you'll ever learn:
  - **man** (short for “*manual*”)
  - **info**
- They help you find information about other commands
  - **man** <cmd> or **info** <cmd> retrieves detailed information about <cmd>
  - **man -k** <keyword> searches the man page summaries (faster, and will probably give better results)
  - **man -K** <keyword> searches the full text of the man pages
- **command --help**
  - [isaac@saw377 ~]\$ ls --help
  - Usage: ls [OPTION]... [FILE]...
  - List information about the FILES (the current directory by default).
  - Sort entries alphabetically if none of -cftuvSUX nor --sort...

# Some Special File Names

- Some file names are special:
  - `/` The root directory (not to be confused with the root user)
  - `.` The current directory
  - `..` The parent (previous) directory
  - `~` My home directory
- Examples:
  - `./a` same as `a`
  - `../isaac/x` go up one level then look in directory `isaac` for `x`

# Command for Directories

- **ls**
  - **LiSts** the contents of the specified directories (or the current directory if no files are specified)
  - Syntax: `ls [<file> ... ]`
  - Example: `ls backups`
- **pwd**
  - shows the present directory info
  - **Print Working Directory**
- **cd**
  - **Change Directory** (or your home directory if unspecified)
  - Syntax: `cd <directory>`
  - Examples:
    - `cd backups/unix-tutorial`
    - `cd ../class-notes`

## (cont'd)

- **mkdir**
  - **MaKe DIRectory**
  - Syntax: `mkdir <directories>`
  - Example: `mkdir backups class-notes`
- **rmdir**
  - **ReMove DIRectory**, which *must be empty*
  - Syntax: `rmdir <directories>`
  - Example: `rmdir backups class-notes`

# Files

- Unlike Windows, in Unix file types (e.g. “executable files,” “data files,” “text files”) are *not* determined by file extension (e.g. “foo.exe”, “foo.dat”, “foo.txt”)
- Thus, the file-manipulation commands are few and simple
- Many commands only use 2 letters
- rm
  - **ReMoves** a file, ***without a possibility of “undelete!”***
  - Syntax: rm [options] <file(s)>
  - Example: rm tutorial.txt backups/old.txt
  - -r option: recursive (delete directories)
  - -f option: force. Do no matter what

# Files (cont'd)

- **cp**

- **CoPies** a file, preserving the original
- Syntax: `cp [options] <sources> <destination>`
- Example: `cp tutorial.txt tutorial.txt.bak`
- `-r` option: recursive (copies directories)

- **mv**

- **MoVes** (renames) a file or directory, destroying the original
- Syntax: `mv [options] <sources> <destination>`
- Examples:
  - `mv tutorial.txt tutorial.txt.bak`
  - `mv tutorial.txt tutorial-slides.ppt backups/`


**Note:** Both of these commands will over-write existing files without warning you!



# More Commands

- **diff** - attempts to determine the minimal set of changes needed to convert a file specified by the first argument into the file specified by the second argument
  - Syntax: `diff [options] <FILES>`
  - Example: `diff a.txt a1.txt`
- **find** - Searches a given file hierarchy specified by path, finding files that match the criteria given by expression
  - Syntax: `find [path...] [expression]`
  - Example: `find ./ -name "tes.h" -print`

# Text editor in command line interface



The screenshot shows a terminal window with a macOS-style title bar. The title bar text is 'Isaac — nano — 96x28'. Below the title bar is a black status bar with three sections: 'GNU nano 2.0.6' on the left, 'New Buffer' in the center, and 'Modified' on the right. The main text area of the terminal contains the text 'This is a text editor Nano!' followed by a cursor. At the bottom of the terminal, there is a help menu with two columns of text. The first column lists '^G Get Help' and '^X Exit'. The second column lists '^O WriteOut', '^J Justify', '^R Read File', and '^W Where Is'. The third column lists '^Y Prev Page' and '^V Next Page'. The fourth column lists '^K Cut Text' and '^U UnCut Text'. The fifth column lists '^C Cur Pos' and '^T To Spell'.

```
Isaac — nano — 96x28
GNU nano 2.0.6          New Buffer          Modified
This is a text editor Nano!

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

# File Permissions

- Linux provides three kinds of permissions:
  - Read (r, 4) - users with read permission may read the file or list the directory
  - Write (w, 2) - users with write permission may write to the file or new files to the directory
  - Execute (x, 1) - users with execute permission may execute the file or lookup a specific file within a directory

# File Permissions

- The long version of a file listing (`ls -l`) will display the file permissions:

```
-rwxrwxr-x  1 rvdheij  rvdheij      5224 Dec 30 03:22 hello
-rw-rw-r--  1 rvdheij  rvdheij       221 Dec 30 03:59 hello.c
-rw-rw-r--  1 rvdheij  rvdheij     1514 Dec 30 03:59 hello.s
drwxrwxr-x  7 rvdheij  rvdheij     1024 Dec 31 14:52 posixuft
```

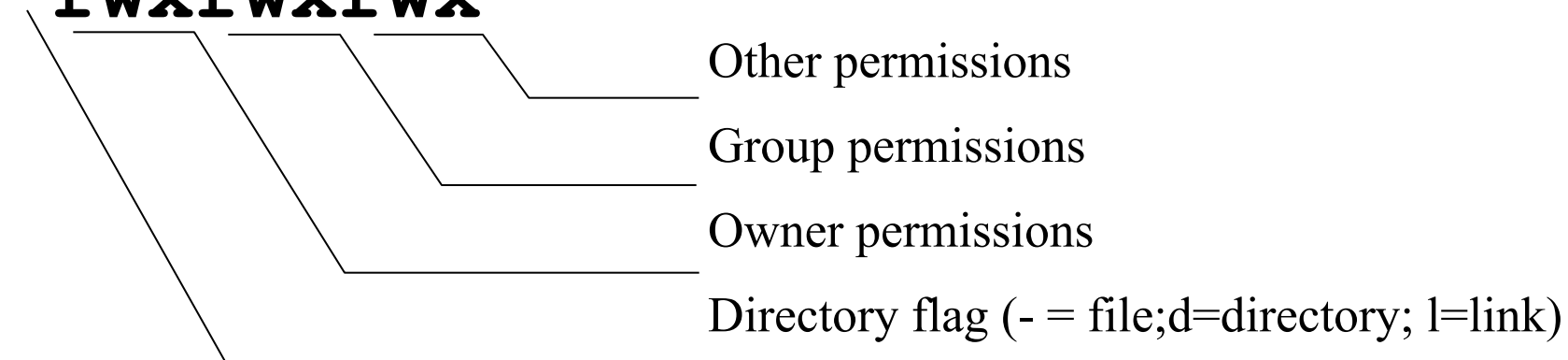
Permissions

Group

Owner

# Interpreting File Permissions

**-rwxrwxrwx**



# Changing File Permissions

- Use the chmod command to change file permissions
  - The permissions are encoded as an octal number

```
chmod 755 file # Owner=rwx Group=r-x Other=r-x
chmod 500 file2 # Owner=r-x Group=--- Other=---
chmod 644 file3 # Owner=rw- Group=r-- Other=r--

chmod +x file # Add execute permission to file for all
chmod o-r file # Remove read permission for others
chmod a+w file # Add write permission for everyone
```

# Processes

- Foreground
  - When a command is executed from the prompt and runs to completion at which time the prompt returns is said to run in the foreground
- Background
  - When a command is executed from the prompt with the token “&” at the end of the command line, the prompt immediately returns while the command continues is said to run in the background
- Check the process
  - Command: ps, top, kill



# Processes

& causes process to be run in  
“background”

```
[root@penguinvm log]# sleep 10h &
```

```
[1] 6718
```

```
[root@penguinvm log]# ps -ef
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	6718	6692	0	14:49	ttyp0	00:00:00	sleep 10h

Job Number

Process ID (ID)

Parent Process ID

# Command for Processes

- kill - sends a signal to a process or process group
- You can only kill your own processes unless you are root

```
UID          PID    PPID    C  STIME TTY          TIME CMD
root         6715    6692    2  14:34 tttyp0       00:00:00 sleep 10h
root         6716    6692    0  14:34 tttyp0       00:00:00 ps -ef
[root@penguinvm log]# kill 6715
[1]+  Terminated                  sleep 10h
```

# Environment Variables

- Environment variables are global settings that control the function of the shell and other Linux programs. They are sometimes referred to as global shell variables.
- Check your environment

```
[isaac@saw377 ~]$ env
MKLROOT=/opt/sharcnet/intel/11.0.083/ifc/mkl
MODULE_VERSION_STACK=3.2.6
MANPATH=/opt/sharcnet/octave/current/share/man:/opt/sharcnet/netcdf/current/man:
FOAM_SOLVERS=/work/isaac/OpenFOAM/OpenFOAM-1.6/applications/solvers
FOAM_APPBIN=/work/isaac/OpenFOAM/OpenFOAM-1.6/applications/bin/linux64GccDPOpt
FOAM_TUTORIALS=/work/isaac/OpenFOAM/OpenFOAM-1.6/tutorials
FOAM_JOB_DIR=/work/isaac/OpenFOAM/jobControl
HOSTNAME=saw377
snrestart=--nosrun /opt/sharcnet/blcr/current/bin/sn_restart.sh
IPPROOT=/opt/sharcnet/intel/11.0.083/icc/ipp/em64t
INTEL_LICENSE_FILE=/opt/sharcnet/intel/11.0.083/ifc/licensesADFBIN=/opt/sharcnet/adf/current/bin
```

# Environment Variables

- Using Environment Variables:
  - `echo $VAR`
  - `cd $VAR`
  - `cd $HOME`
- Displaying - use the following commands:
  - `set` (displays local & env. Vars)
  - `export`
- Vars can be retrieved by a script or a program

# Some Important Environment

- HOME
  - Your home directory (often be abbreviated as “~”)
- TERM
  - The type of terminal you are running (for example vt100, xterm, and ansi)
- PWD
  - Current working directory
- PATH
  - List of directories to search for commands

# PATH Environment Variable

- Controls where commands are found
  - PATH is a list of directory pathnames separated by colons. For example:  
`? PATH=/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/home/alex/bin`
  - If a command does not contain a slash, the shell tries finding the command in each directory in PATH. The first match is the command that will run
  - Set in `/etc/profile`, `~/.profile`, `~/.bashrc`

# Modules

- What is a module system?
  - A user interface to provide for the dynamic modification of a user's environment via modulefiles.

# Modules (Example: loading WRF)

- Module list – list up the presently loaded modules

```
[isaac@hnd50:~] module list
```

```
Currently Loaded Modulefiles:
```

1) moab/5.4.2	7) r/2.10.0	13) gromacs/4.0.5
2) sq-tm/2.4	8) namd/2.7b3	14) vmd/1.8.7
3) intel/11.0.083	9) ansys/12.1.1	15) util/2.0
4) openmpi/intel/1.4.2	10) lsdyna/ls971dR5.0	16) user-environment/1.0.0
5) compile/1.3	11) fftw/intel/2.1.5	
6) octave/3.2.4	12) lammps/10.08.2010	

- Module avail – list up all available modules

```
[isaac@hnd50:~] module avail
```

```
----- /opt/sharcnet/modules -----  
acml/gfortran/3.6.0      acml/pgi-int64/4.2.0      lammps/10.08.2010  
acml/gfortran/4.2.0      openmpi/intel-debug/1.4.3  wrf/3.2
```



# Modules (Cont'd)

- Module show [module] – load the module into the env

```
[isaac@hnd50:~] module show wrf/3.2
```

```
-----  
/opt/sharcnet/modules/wrf/3.2:
```

```
module-whatism Provide WRF/WPS 3.2 built using intel 11.0.083 and openmpi 1.4.2 on centos.
```

```
conflict      wrf
```

```
prereq        intel/11.0.083
```

```
prereq        openmpi/intel/1.4.2
```

```
module       load gmp/4.3.2
```

```
module       load mpfr/2.4.2
```

```
module       load netcdf/intel/4.1.2
```

```
prepend-path  PATH /opt/sharcnet/wrf/3.2/wrfv3/main:/opt/sharcnet/wrf/3.2/wrfv3/run:/opt/  
sharcnet/wrf/3.2/wrfv3/tools:/opt/sharcnet/wrf/3.2/wps:/opt/sharcnet/wrf/3.2/wps/util
```

```
prepend-path  LD_RUN_PATH /opt/sharcnet/wrf/3.2/wps_libs/lib
```

```
prepend-path  --delim  LDFLAGS -L/opt/sharcnet/wrf/3.2/wps_libs/lib -L/opt/sharcnet/wrf/3.2/  
wrfv3/main
```

```
prepend-path  --delim  CPPFLAGS -I/opt/sharcnet/wrf/3.2/wps_libs/include -I/opt/sharcnet/wrf/  
3.2/wrfv3/inc
```

# Modules (Cont'd)

- Module load [module] – load the module into the env

```
[isaac@hnd50:~] module load wrf/3.2
```

```
[isaac@hnd50:~] module list
```

Currently Loaded Modulefiles:

- |                        |                       |                       |                               |
|------------------------|-----------------------|-----------------------|-------------------------------|
| 1) moab/5.4.2          | 6) octave/3.2.4       | 11) fftw/intel/2.1.5  | 16) user-environment/1.0.0    |
| 2) sq-tm/2.4           | 7) r/2.10.0           | 12) lammps/10.08.2010 | <b>17) gmp/4.3.2</b>          |
| 3) intel/11.0.083      | 8) namd/2.7b3         | 13) gromacs/4.0.5     | <b>18) mpfr/2.4.2</b>         |
| 4) openmpi/intel/1.4.2 | 9) ansys/12.1.1       | 14) vmd/1.8.7         | <b>19) netcdf/intel/4.1.2</b> |
| 5) compile/1.3         | 10) lsdyna/ls971dR5.0 | 15) util/2.0          | <b>20) wrf/3.2</b>            |

# Modules (Cont'd)

- Module unload [module] – unload the module from the env

```
[isaac@hnd50:~] module unload wrf
```

```
[isaac@hnd50:~] module list
```

Currently Loaded Modulefiles:

1) moab/5.4.2	5) compile/1.3	9) ansys/12.1.1	13) gromacs/4.0.5
2) sq-tm/2.4	6) octave/3.2.4	10) lsdyna/ls971dR5.0	14) vmd/1.8.7
3) intel/11.0.083	7) r/2.10.0	11) fftw/intel/2.1.5	15) util/2.0
4) openmpi/intel/1.4.2	8) namd/2.7b3	12) lammps/10.08.2010	16) user-

environment/1.0.0

# Editing Text

- Which text editor is “the best” is a holy war. Pick one and get comfortable with it.
- Three text editors you should be aware of:
  - nano – An improved ‘pico’ editor
    - To quit: Ctrl-x
  - emacs/xemacs – A heavily-featured editor commonly used in programming
    - To quit: Ctrl-x Ctrl-c
  - vim/vi – Another editor, also used in programming
    - To quit: <Esc> : q <Enter> (or QQ -- capitals matter)

Knowing the basics of emacs and vim will help with the rest of Unix; many programs have similar key sequences.

# Pipe and Redirection

## □ Redirection (< or >)

% ls -l > lsoutput.txt (save output to lsoutput.txt)

% ps >> lsoutput.txt (append to lsoutput.txt)

% more < killout.txt (use killout.txt as parameter to more)

## □ Pipe (|)

▣ Process are executed concurrently

% ps | sort | more

% ps -xo comm | sort | uniq | grep -v sh | more

% cat mydata.txt | sort | uniq | > mydata.txt (generates an empty file !)

# 'for' Loops

```
for var in list
do
    statements
done
```

- statements are executed with `var` set to each value in the list.

```
#!/bin/bash
let sum=0
for num in 1 2 3 4 5
do
    let "sum = $sum + $num"
done
echo $sum
```

→ {1..5}

# For-loop like a C-program

```
#!/bin/bash
echo -n "Enter a number : "
read num
let sum=0
for(( i=1; $i<$num+1;i=$i+1));
do
    let sum=$sum+$i
done
echo "the sum of the first $num numbers is $sum"
```

# Conditional statement

```
#!/bin/bash

if [ expression ];
then
    statements
elif [ expression ];
then
    statements
else
    statements
fi

if [ "$name" = "$USER" ];
then
    echo "Hello, $name. How are you today ?"
else
    echo "You are not $USER, so who are you ?"
fi
```