Introduction to **Linux**

Augustine Abaris
Research Computing Services
Information Services & Technology

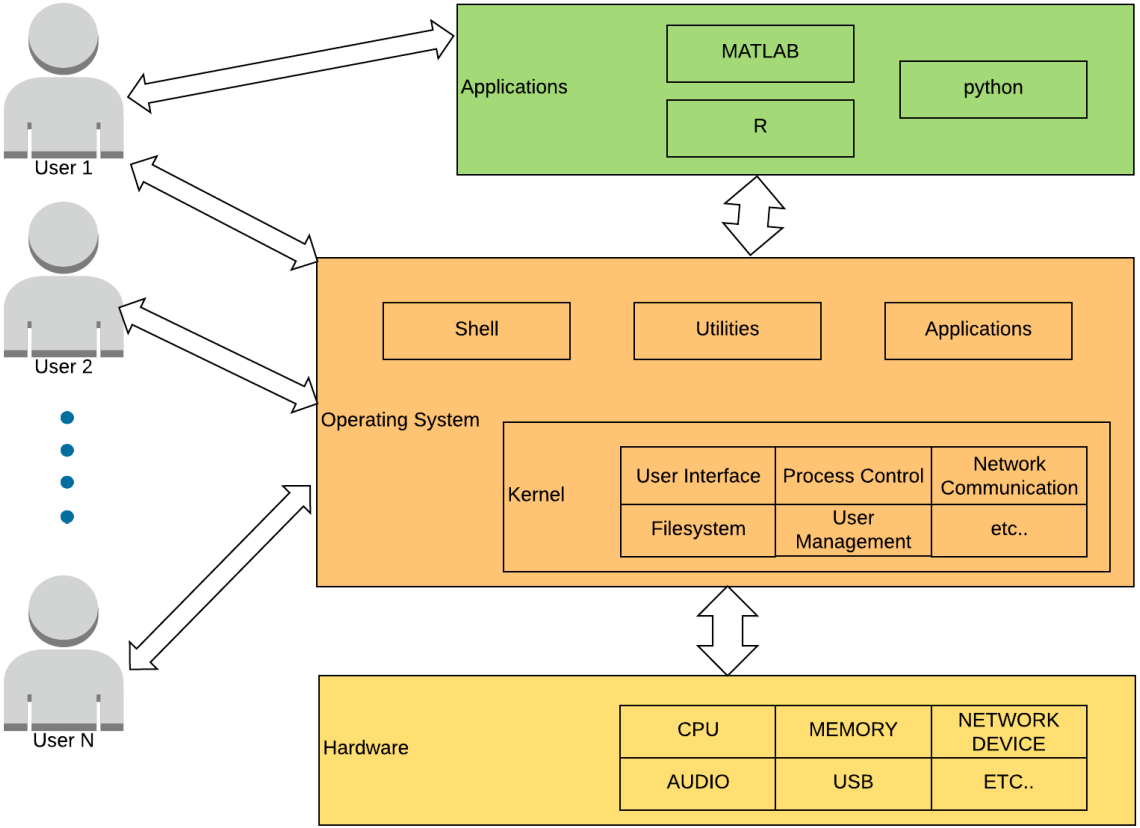
BOSTON UNIVERSITY

Topics for Today

- Research Computing Services
- Linux Overview
- Linux Interaction - Shell and Commands
- I/O redirection (pipes, etc.)
- Navigating the file system
- Processes and job control
- Editors
- Creating and Running Code

Linux

What is an operating system?



What is Linux?

- Operating System
- Originated in early-90s
- Free
- Open Source

Where is Linux?

- Desktop computers and laptops
- Servers
- High performance clusters
- Embeded systems
- Home entertainment (smart TVs, IoT devices)
- Cellphones (Android)

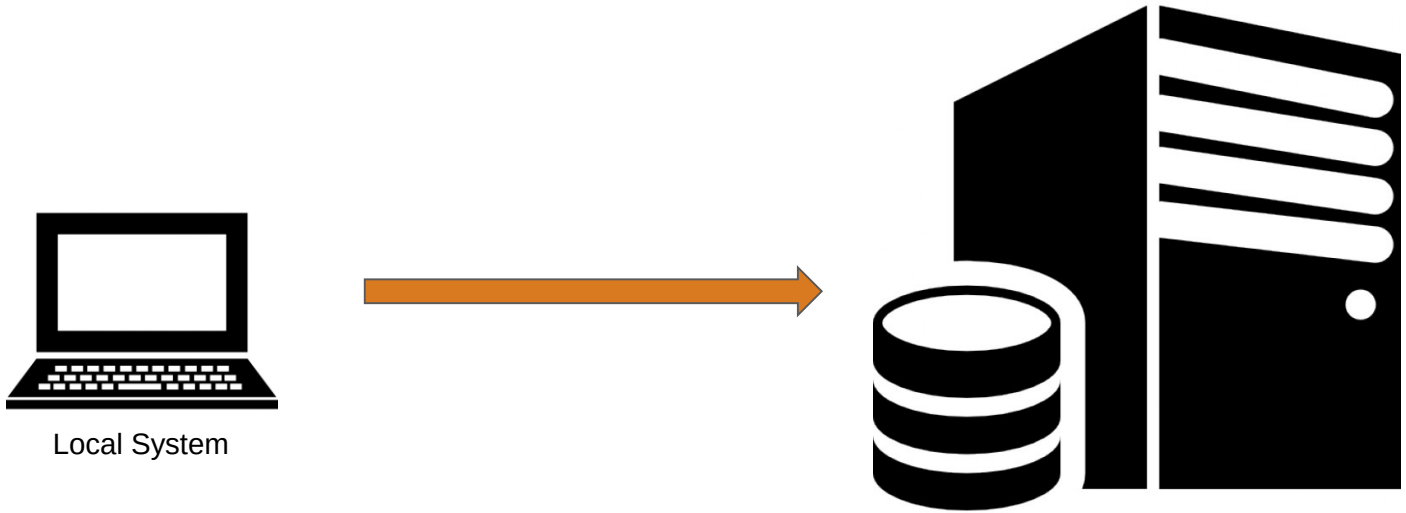
Why Linux

- Free and open-source.
- Powerful for research datacenters
- Personal for desktops and phones
- Universal
- Community (and business) driven.

The most common OS used by BU researchers when working on a server or computer cluster

Connecting

Let’s use Linux

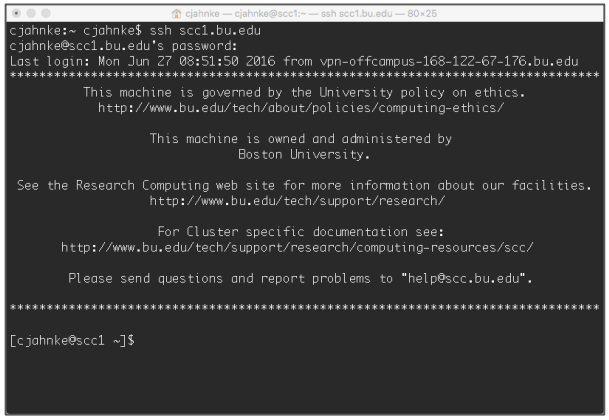

Local System
Remote Server

Connection Protocols and Software

Remote Connections:

Secure **SH**ell

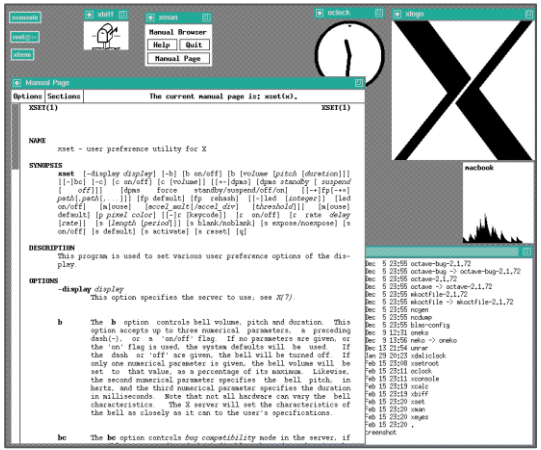
(SSH)



Remote Graphics:

X-Windowing

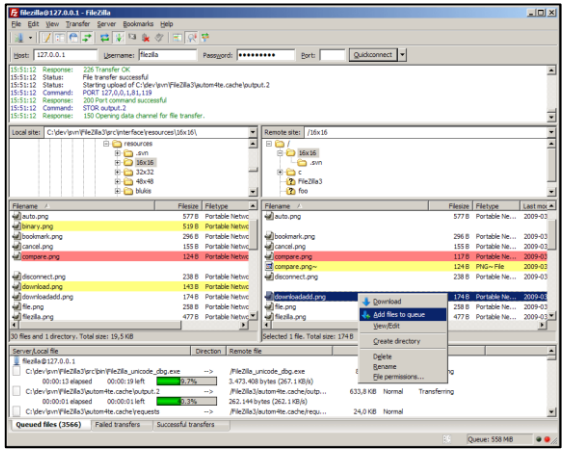
(X, X-Win)



Data Transfer:

Secure **F**ile **T**ransfer **P**rotocol

(SFTP)



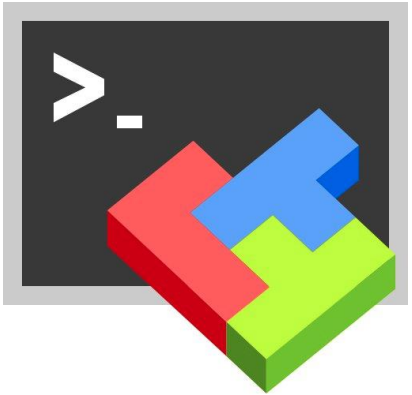
Other protocols too, but let's start with these.

Connecting from Different Platforms

	SSH	X-Win	SFTP
Microsoft Windows	◆———	**MobaXterm** https://mobaxterm.mobatek.net	———◆
Apple macOS	**Terminal** (Built in)	**XQuartz** https://www.xquartz.org	**Cyberduck** https://cyberduck.io
Linux	**Terminal** (Built in)	**X11** (Built in)	**Various** (Built in)

SCC Help: http://www.bu.edu/tech/support/research/system-usage/getting-started

Microsoft Windows



You need software that emulates an “X” terminal and that connects using the “SSH” Secure Shell protocol.

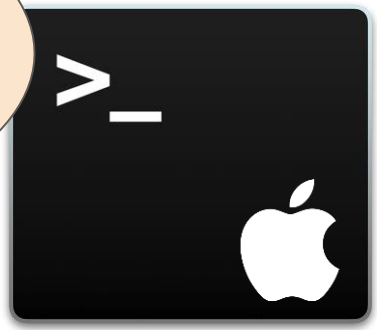
- **Recommended**: MobaXterm
  - Download: http://mobaxterm.mobatek.net/

- Alternatives:
  - SSH/X-Windows: X-Win32
    https://www.bu.edu/tech/services/support/desktop/distribution/xwindows/
  - SFTP: Filezilla
    https://filezilla-project.org/

SCC Help: http://www.bu.edu/tech/support/research/system-usage/getting-started/connect-ssh

Apple macOS

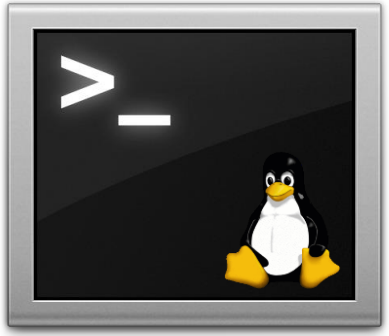
Built in!
Apple macOS is built on Darwin -- a derivative of 4.4BSD-Lite2 and FreeBSD



- SSH: Terminal
  - Built in to macOS
    Applications > Utilities > Terminal
- X-Windows: XQuartz
  - Download: https://www.xquartz.org/
  - Note: This install requires a logout.
- SFTP: Your choice
  - Filezilla: https://filezilla-project.org/ (Cross-platform, open-source)
  - Cyberduck: https://cyberduck.io (macOS native, drag-and-drop)
  - Many others

SCC Help: http://www.bu.edu/tech/support/research/system-usage/getting-started/connect-ssh

Linux



- SSH: Terminal
  - Built in to Linux
    Applications > System > Terminal
- X-Windows: X11
  - Built in to Linux
  - Use your package manager.
- SFTP: Your choice
  - Usually has one Built in.
  - Alternate: Filezilla (https://filezilla-project.org/)

SCC Help: http://www.bu.edu/tech/support/research/system-usage/getting-started/connect-ssh

Connecting

- Use your Shared Computing Cluster account if you have one.

- Tutorial accounts if you need one.
  - Username:
  - Password:

Tutorial credentials blocked for print.
This box disappears during presentation

```
[local_prompt]$ ssh username@scc1.bu.edu
```

Get supplementary files

- At the command prompt, type the following:

```
[username@scc1 ~]$ cd ~
[username@scc1 ~]$ tar xf /scratch/linux-materials.tar
[username@scc1 ~]$ ls
c   data   haystack   scripts
```

Linux Interaction

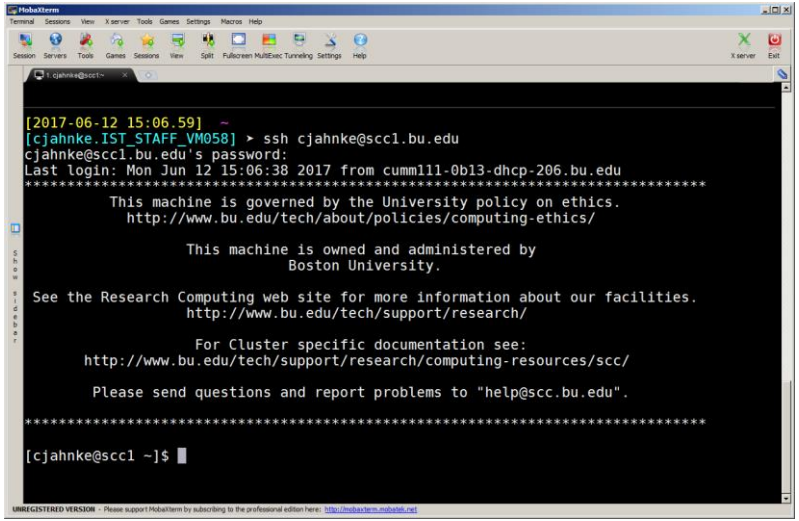
Shell, Prompt, Commands and System Use

Linux: The Shell

- Program that ***interprets commands*** and sends them to the OS

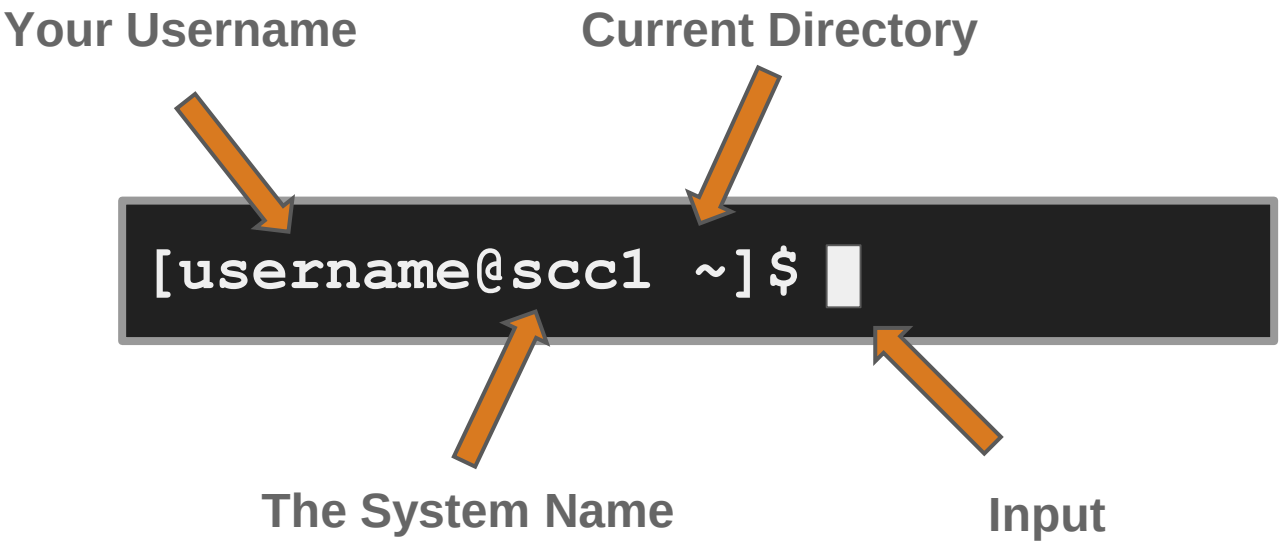
- Provides:
  - Built-in commands
  - Programming control structures
  - Environment variables

- Linux supports multiple shells.
  - The default on SCC is **`Bash`**.



"**`Bash`**" = "**B**ourne-**a**gain **Sh**ell"
(GNU version of ~1977 shell written by Stephen Bourne)

Linux: The “prompt”



(In Linux “ ~ ” is a shorthand for your home directory.)

Linux: Command Basics

```
[username@scc1 ~]$ command --option argument
```

- **Command**: Command/program that does one thing
- **Options**: Change the way a command does that one thing
  - Short form: Single-dash and one letter e.g. **`ls -a`**
  - Long form: Double-dash and a word e.g. **`ls --all`**
- **Argument**: Provides the input/output that the command interacts with.

For more information about any command, use **`man`** or **`info`** (e.g. "**`man ls`**")

Commands: Hands-On

- After you connect, type
  - `whoami` # my login
  - `hostname` # name of this computer
  - `echo “Hello, world”` # print characters to screen
  - `echo $HOME` # print environment variable
  - `echo my login is $(whoami)` # replace $(xx) with program output
  - `date` # print current time/date
  - `cal` # print this month’s calendar
  - `shazam` # bad command

Commands: Hands-On Options

- Commands have three parts; command, options and arguments/parameters.

  Example: cal –j 3 1999. “cal” is the command, “-j” is an option (or switch), “3” and “1999” are arguments/parameters.

```
[username@scc1 ~]$ cal -j 3 1999
```

- What is the nature of the prompt?
- What was the system’s response to the command?

Commands

“Small programs that do one thing well”

- The Unix Programming Environment, Kernighan and Pike

  … at its heart is the idea that the power of a system comes more from the relationships among programs than from the programs themselves. Many UNIX programs do quite trivial things in isolation, but, combined with other programs, become general and useful tools.

Commands: Selected text processing utilities

- **`awk`** Pattern scanning and processing language
- **`cat`** Display file(s)
- **`cut`** Extract selected fields of each line of a file
- **`diff`** Compare two files
- **`grep`** Search text for a pattern
- **`head`** Display the first part of files
- **`less`** Display files on a page-by-page basis
- **`sed`** Stream editor (esp. search and replace)
- **`sort`** Sort text files
- **`split`** Split files
- **`tail`** Display the last part of a file
- **`tr`** Translate/delete characters
- **`uniq`** Filter out repeated lines in a file
- **`wc`** Line, word and character count

Just a few of the commands for text processing

Variables and Environment Variables

- *Variables* are named storage locations.
  - **`USER=augustin`**
  - **`foo=“this is foo’s value”`**

- “*Environment variables*” are variables used and *shared* by the shell
  - For example, **`$PATH`** tells the system where to find commands.

- Environment variables are shared with programs that the shell runs.

Bash variables

- To create a new variable, use the assignment operator ‘=‘

```
[username@scc1 ~]$ foo=“this is foo’s value”
```

- The foo variable can be printed with **echo**

```
[username@scc1 ~]$ echo $foo
this is foo’s value
```

- To make **$foo** visible to programs run by the shell (i.e., make it an “environment variable”), use **export**:

```
[username@scc1 ~]$ export foo
```

Environment Variables

- To see all currently defined environment variable, use **`printenv`**:

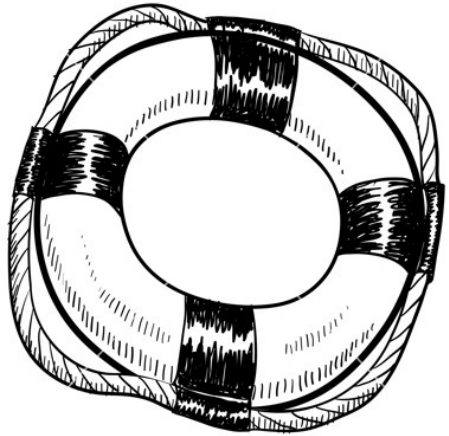
```
[username@scc1 ~]$ printenv
HOSTNAME=scc1
TERM=xterm-256color
SHELL=/bin/bash
HISTSIZE=1000
TMPDIR=/scratch
SSH_CLIENT=168.122.9.131 37606 22
SSH_TTY=/dev/pts/191
USER=cjahnke
MAIL=/var/spool/mail/cjahnke
PATH=/usr3/bustaff/cjahnke/apps/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbi
n
PWD=/usr3/bustaff/cjahnke/linux-materials
LANG=C
MODULEPATH=/share/module/bioinformatics:/share/module/chemistry
SGE_ROOT=/usr/local/ogs-ge2011.11.p1/sge_root
HOME=/usr3/bustaff/cjahnke
```

Command History and Command Line Editing

- Try the **`history`** command
- Choose from the command history using the up ↑ and down ↓ arrows
- To redo your last command, try **`!!`**
- To go further back in the command history try **`!`**, then the number as shown by history (e.g., **`!132`**). Or, !ls, for example, to match the most recent ‘ls’ command.
- What do the left ← and right → arrow do on the command line?
- Try the **`<Del>`** and **`<Backspace>`** keys

Help with Commands

- Type
  - **`date --help`**
  - **`man date`**
  - **`info date`**
- BASH built-ins
  - A little different from other commands
  - Just type the command **`‘help’`**
  - Or **`‘man bash’`**



Yes, you can always Google it.

On using **‘man’** with **‘less’**

- The **man** command outputs to a pager called **less**, which supports many ways of scrolling through text:
  - **Space, f** # page forward
  - **b** # page backward
  - **<** # go to first line of file
  - **>** # go to last line of file
  - **/** # search forward (n to repeat)
  - **?** # search backward (N to repeat)
  - **h** # display help
  - **q** # quit help

I/O Redirection

I/O redirection with pipes

- Many Linux commands print to “standard output”, which defaults to the terminal screen. The ‘ | ’ (pipe) character can be used to divert or “redirect” output to another program or filter.
  - `w` # show who’s logged on
  - `w | less` # pipe into the ‘less’ pager
  - `w | grep ‘tuta’` # pipe into grep, print lines containing ‘tuta’
  - `w | grep –v ‘tuta’` # print only lines not containing ‘tuta’
  - `w | grep ‘tuta’ | sed s/tuta/scholar/g` # replace all ‘tuta’ with ‘scholar’

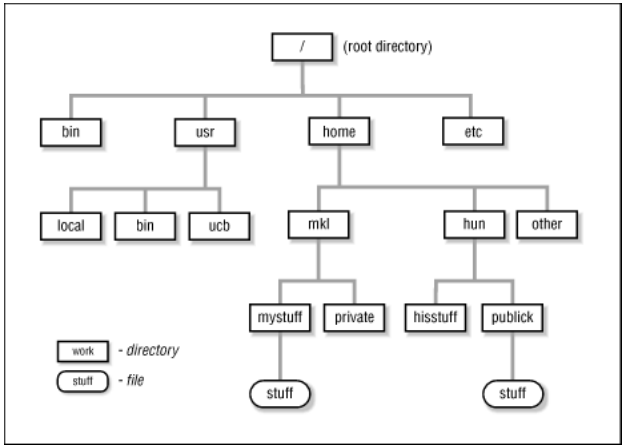
More examples of I/O redirection

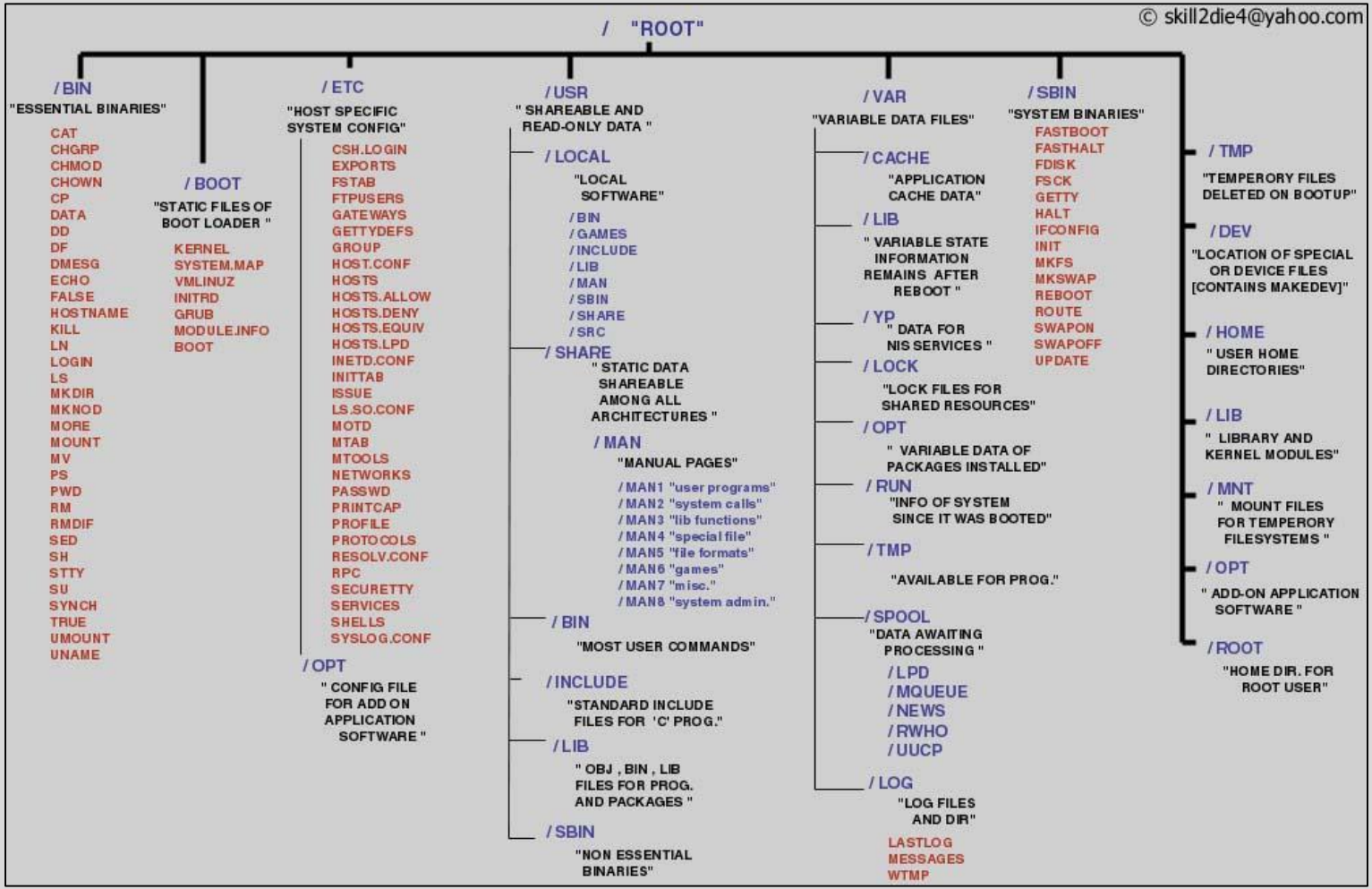
- Try the following (use up arrow to avoid retyping each line):
  - **`w | wc`**
    # count lines
  - **`w | cut –d ‘ ’ –f1 | sort`** # sort users
  - **`w | cut –d ‘ ’ –f1 | sort | uniq`** # eliminate duplicates
- We can also redirect output into a file:
  - **`w | cut –d ‘ ’ –f1 | sort | uniq > users`**
- Note that ‘awk’ can be used instead of ‘cut’:
  - **`w | awk ‘{print $1;}’ | sort | uniq > users`**
- Quiz:

The Filesystem

The Linux File System

- The structure resembles an upside-down tree
- Directories (a.k.a. folders) are collections of files and other directories.
- Every directory has a parent except for the root directory.
- Many directories have subdirectories.




© skill2die4@yahoo.com
/ "ROOT"
/BIN
"ESSENTIAL BINARIES"
CAT
CHGRP
CHMOD
CHOWN
CP
DATA
DD
DF
DMESG
ECHO
FALSE
HOSTNAME
KILL
LN
LOGIN
LS
MKDIR
MKNOD
MORE
MOUNT
MV
PS
PWD
RM
RMDIF
SED
SH
STTY
SU
SYNCH
TRUE
UMOUNT
UNAME
/BOOT
"STATIC FILES OF BOOT LOADER "
KERNEL
SYSTEM.MAP
VMLINUZ
INITRD
GRUB
MODULE.INFO
BOOT
/ETC
"HOST SPECIFIC SYSTEM CONFIG"
CSH.LOGIN
EXPORTS
FSTAB
FTPUSERS
GATEWAYS
GETTYDEFS
GROUP
HOST.CONF
HOSTS
HOSTS.ALLOW
HOSTS.DENY
HOSTS.EQUIV
HOSTS.LPD
INETD.CONF
INITTAB
ISSUE
LS.SO.CONF
MOTD
MTAB
MTOOLS
NETWORKS
PASSWD
PRINTCAP
PROFILE
PROTOCOLS
RESOLV.CONF
RPC
SECURETTY
SERVICES
SHELLS
SYSLOG.CONF
/OPT
" CONFIG FILE FOR ADD ON APPLICATION SOFTWARE "
/USR
" SHAREABLE AND READ-ONLY DATA "
/LOCAL
"LOCAL SOFTWARE"
/BIN
/GAMES
/INCLUDE
/LIB
/MAN
/SBIN
/SHARE
/SRC
/SHARE
" STATIC DATA SHAREABLE AMONG ALL ARCHITECTURES "
/MAN
"MANUAL PAGES"
/MAN1 "user programs"
/MAN2 "system calls"
/MAN3 "lib functions"
/MAN4 "special file"
/MAN5 "file formats"
/MAN6 "games"
/MAN7 "misc."
/MAN8 "system admin."
/BIN
"MOST USER COMMANDS"
/INCLUDE
"STANDARD INCLUDE FILES FOR 'C' PROG."
/LIB
" OBJ , BIN , LIB FILES FOR PROG. AND PACKAGES "
/SBIN
"NON ESSENTIAL BINARIES"
/VAR
"VARIABLE DATA FILES"
/CACHE
"APPLICATION CACHE DATA"
/LIB
" VARIABLE STATE INFORMATION REMAINS AFTER REBOOT "
/YP
" DATA FOR NIS SERVICES "
/LOCK
"LOCK FILES FOR SHARED RESOURCES"
/OPT
" VARIABLE DATA OF PACKAGES INSTALLED"
/RUN
"INFO OF SYSTEM SINCE IT WAS BOOTED"
/TMP
"AVAILABLE FOR PROG."
/SPOOL
"DATA AWAITING PROCESSING "
/LPD
/MQUEUE
/NEWS
/RWHO
/UUCP
/LOG
"LOG FILES AND DIR"
LASTLOG
MESSAGES
WTMP
/SBIN
"SYSTEM BINARIES"
FASTBOOT
FASTHALT
FDISK
FSCK
GETTY
HALT
IFCONFIG
INIT
MKFS
MKSWAP
REBOOT
ROUTE
SWAPON
SWAPOFF
UPDATE
/TMP
"TEMPERORY FILES DELETED ON BOOTUP"
/DEV
"LOCATION OF SPECIAL OR DEVICE FILES [CONTAINS MAKEDEV]"
/HOME
" USER HOME DIRECTORIES"
/LIB
" LIBRARY AND KERNEL MODULES"
/MNT
" MOUNT FILES FOR TEMPERORY FILESYSTEMS "
/OPT
" ADD-ON APPLICATION SOFTWARE "
/ROOT
"HOME DIR. FOR ROOT USER"

Navigating the File System

- Essential navigation commands:
  - **`pwd`** print current directory
  - **`ls`** list files
  - **`cd`** change directory

Navigating the File System

We use pathnames to refer to files and directories in the Linux file system.

- There are two types of pathnames:
  - **Absolute** – The full path to a directory or file; begins with /
  - **Relative** – A partial path that is relative to the current working directory; does not begin with /

Navigating the File System

- Special characters interpreted by the shell for filename expansion:
  - **`~`** your home directory (e.g., /usr1/tutorial/tuta1)
  - **`.`** current directory
  - **`..`** parent directory
  - **`*`** wildcard matching any filename
  - **`?`** wildcard matching any character
  - **`TAB`** try to complete (partially typed) filename

Navigating the File System

- Examples:
  - **`cd /usr/local`** Change directory to /usr/local/lib
  - **`cd ~`** Change to home directory (could just type 'cd')
  - **`pwd`** Print working (current) directory
  - **`cd ..`** Change directory to the "parent" directory
  - **`cd /`** Change directory to the "root"
  - **`ls -d pro*`** Listing of only the directories starting with "pro"

The ls Command

- Useful options for the “**`ls`**” command:
  - **`ls -a`** List all files, including hidden files beginning with a “**`.`**”
  - **`ls -ld *`** List details about a directory and not its contents
  - **`ls -F`** Put an indicator character at the end of each name
  - **`ls -l`** Simple long listing
  - **`ls -lR`** Recursive long listing
  - **`ls -lh`** Give human readable file sizes
  - **`ls -lS`** Sort files by file size
  - **`ls -lt`** Sort files by modification time (very useful!)

Some Useful File Commands

- **`cp [file1] [file2]`** copy file
- **`mkdir [name]`** make directory
- **`rmdir [name]`** remove (empty) directory
- **`mv [file] [destination]`** move/rename file
- **`rm [file]`** remove (-r for recursive)
- **`file [file]`** identify file type
- **`less [file]`** page through file
- **`head -n N [file]`** display first ***N*** lines
- **`tail -n N [file]`** display last ***N*** lines
- **`ln –s [file] [new]`** create symbolic link
- **`cat [file] [file2…]`** display file(s)
- **`tac [file] [file2…]`** display file in reverse order
- **`touch [file]`** update modification time
- **`od [file]`** display file contents, esp. binary

Manipulating files and directories

- Examples:
  - `cd`
    # The same as `cd ~`
  - `mkdir test`
  - `cd test`
  - `echo 'Hello everyone' > myfile.txt`
  - `echo 'Goodbye all' >> myfile.txt`
  - `less myfile.txt`
  - `mkdir subdir1/subdir2` # Fails. Why?
  - `mkdir -p subdir1/subdir2` # Succeeds
  - `mv myfile.txt subdir1/subdir2`
  - `cd ..`
  - `rmdir test`
    # Fails. Why?
  - `rm -rf test`

Finding a needle in a haystack

- The **`find`** command has a rather unfriendly syntax, but can be exceedingly helpful for locating files in heavily nested directories.

- Examples:
  - `find ~ -name bu -type d` # search for “bu” directories in **~**
  - `find . -name my-file.txt` # search for my-file.txt in .
  - `find ~ -name '*.txt'` # search for “***.txt**” in **~**

- Quiz:
  - Can you use **`find`** to locate a file called “needle” in your haystack directory?
  - Extra credit: what are the contents of the “needle” file?

Processes & Job Control

Processes and Job Control

- As we interact with Linux, we create numbered instances of running programs called “processes.” You can use the ‘ps’ command to see a listing of your processes (and others!). To see a long listing, for example, of all processes on the system try:

```
[username@scc1 ~]$ ps -ef
```

- To see all the processes owned by you and other members of the class, try:

```
[username@scc1 ~]$ ps -ef | grep tuta
```

Processes and job control

- Use “`top`” to see active processes.

```
Tasks: 408 total,   1 running, 407 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.3%us,  0.1%sy,  0.0%ni, 99.6%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  99022756k total, 69709936k used, 29312820k free,   525544k buffers
Swap:  8388604k total,        0k used,  8388604k free, 65896792k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 7019 root      20   0  329m 137m 4852 S  4.0  0.1 217:01.56 sge_qmaster
38246 isw       20   0 88724 2764 1656 S  0.7  0.0   0:01.28 sshd
41113 cjahnke   20   0 13672 1512  948 R  0.7  0.0   0:00.03 top
 2324 root      20   0     0    0    0 S  0.3  0.0   0:21.82 kondemand/2
 7107 nobody    20   0 89572  10m 2400 S  0.3  0.0   2:18.05 gmond
27409 theavey   20   0 26652 1380  880 S  0.3  0.0   0:34.84 tmux
    1 root      20   0 25680 1604 1280 S  0.0  0.0   0:05.74 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.07 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.89 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:01.72 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 stopper/0
```

(refreshes every 2 seconds)

Foreground/background

- Thus far, we have run commands at the prompt and waited for them to complete. We call this running in the "**foreground**."
- Use the "**&**" operator, to run programs in the "**background**",
  - Prompt returns immediately without waiting for the command to complete:

```
[username@scc1 ~]$ mycommand &
[1] 54356                                   ← process id
[username@scc1 ~]$
```

Process Control Practice

- Let’s look at the “countdown” script, in your scripts folder for practice

```
[username@scc1 ~]$ cd ~/scripts
[username@scc1 ~]$ cat countdown
```

- Make the script executable with chmod:

```
[username@scc1 ~]$ chmod +x countdown
```

- First, run it for a few seconds, then kill with Control-C.

```
[username@scc1 ~]$ ./countdown 100
100
99
98
^C          ← Ctrl-C  = (^C)
```

Process control

- Now, let’s try running it in the background with &:

```
[username@scc1 ~]$ ./countdown 60 &
[1] 54355
[username@scc1 ~]$
60
59
```

- The program’s output is distracting, so redirect it to a file:

```
[username@scc1 ~]$ countdown 60 > c.txt &
[1] 54356
[username@scc1 ~]$
```

Process control

- Type ‘**`ps`**’ to see your countdown process.
- Also, try running ‘jobs’ to see any jobs running in the background from this bash shell.
- To kill the job, use the ‘kill’ command, either with the five-digit process id:
  - **`kill 54356`**
- Or, you can use the job number (use ‘jobs’ to see list) with ‘%’:
  - **`kill %1`**

Backgrounding a running job with C-z and ‘bg’

- Sometimes you start a program, then decide to run it in the background.

```
[username@scc1 scripts]$ ./countdown 200 > c.out
^Z                                                  ← Ctrl-Z  = (^Z)
[1]+  Stopped                 ./countdown 200 > c.out

[username@scc1 scripts]$ bg
[1]+ ./countdown 200 > c.out &

[username@scc1 scripts]$ jobs
[1]+  Running                 ./countdown 200 > c.out &

[username@scc1 scripts]$
```

Editors

File Editors

- gedit
  - Notepad-like editor with some programming features (e.g., syntax highlighting). Requires X-Windows.
- nano
  - Lightweight editor. Non-Xwindows.
- emacs
  - Swiss-army knife, has modes for all major languages, and can be customized. Formerly steep learning curve has been reduced with introduction of menu and tool bars. Can be used under Xwindows or not.
- vim
  - A better version of ‘vi’ (an early full-screen editor). Very fast, efficient. Steep learning curve. Popular among systems programmers. Terminal or X-Windows.

“Hello, world” in C

- **`cd`** to “**`~/c`**”, and read hello.c into your editor of choice.
- Modify the text on the printf line between “**`[`**“ and “**`]`**” and save the file.
- Produce an executable file called “hello” by compiling the program with gcc:

```
[username@scc1 ~]$ gcc –o hello hello.c
```

- Run the program at the command line:

```
[username@scc1 ~]$ ./hello
```

- Optional: modify countdown script to run hello program

Questions?