Introduction to Linux/UNIX (draft)

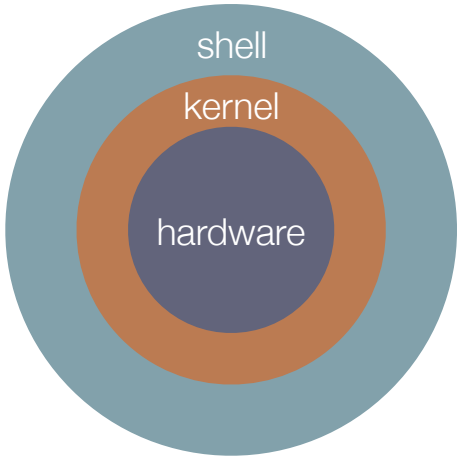
This slide-set will be for all the Linux/Unix materials Sep 5, 7, 12 2017. It is not complete yet, but I wanted to give you an idea of what to expect next week

Introduction to Unix/Linux

- There is more material in this presentation than we can cover in detail, please use it as a reference (along with online resources and examples)

Unix/Linux OS

- Unix/Linux has a kernel and one or more shells
- The shell is the command interpreter (ie the program that processes the command you enter in your terminal emulator (or in a process shell script).
- The kernel sits on top of the hardware and is the core of the OS; it receives tasks from the shell and performs them.



UNIX/Linux: Shell Flavors

There are two main ‘flavors’ of shells. We will be assuming the bash shell.

- Bourne created what is now known as the standard shell: “sh” or “bourne shell”. Its syntax roughly resembles Pascal. Its derivatives include “ksh” (“korn shell”) and now, the most widely used “bash” (“bourne again shell”)

- One of the creators of the C language implemented a shell to have a C like syntax “csh” “C-shell” Most people use the “tcsh” variant.

Hands-on: Login to class computers

- Use your EID's to login to the class computers
  - Open ssh client (putty on the lab machines) or just Terminal on Mac
- Use your TACC username and password to ssh to isp.tacc.utexas.edu

Hands on: Introduction to UNIX/LINUX

1. **Very Basic Commands/Interacting with the shell** (remind me to give you the printed“cheat-sheet” https://files.fosswire.com/2007/08/fwunixref.pdf or I like this online cheatsheet too http://www.rain.org/~mkummel/unix.html

2. File attributes and permissions

3. Job control

4. UNIX/Linux Environment variables

5. Text Editors

Hands-on: pwd, ls

- When you first login, you current working directory is your home directory. Your home directory has the same name as your user-name, and it is where your personal files and subdirectories are saved.
  Find out your current directory type:

```
$ pwd
```

- List what is in your directory type:

```
$ ls
```

`ls` lists only those ones whose name does not begin with a dot (.) Files beginning with a dot (.) are known as hidden files and usually contain important program configuration information.

List all the files: Note -a is one of the command options. There are many more

```
$ ls -a
```

Note -a is one of the command options. There are many more. Find out more about any commands (including usage and options) use man

Hands-on: man

Note -a is one of the command options. There are many more. Find out more about any commands (including usage and options) use man

```
$ man ls
$ man man
```

Hands-on: mkdir, cd

- Navigate to the parent directory you can use two dots:

```
$ cd ..
```

- You can navigate up the hierarchy using successive ../../../

```
$ ls ../../
$ cd ../../
```

- or the full path up the hierarchy

```
$ cd /home/adb/unix_tutorial
$ pwd
$ ls
```

If you are following along you are in /home and when you ls you can see everyone else home directory.

Hands-on: mkdir, cd

- Create a subdirectory in your home directory:

```
$ mkdir unix_tutorial
```

- See what you have created:

```
$ ls
```

- Go into that directory:

```
$ cd unix_tutorial
```

- See where you are:

```
$ pwd
```

- Create another subdirectory called “backups” and change to that directory. Check your directory structure with “pwd” and you should see:

```
[adb@isp02 backups]$ pwd
/home/adb/unix_tutorial/backups
```

Hands-on: practice with cd to navigate the file system

- Navigate to the parent directory you can use two dots:

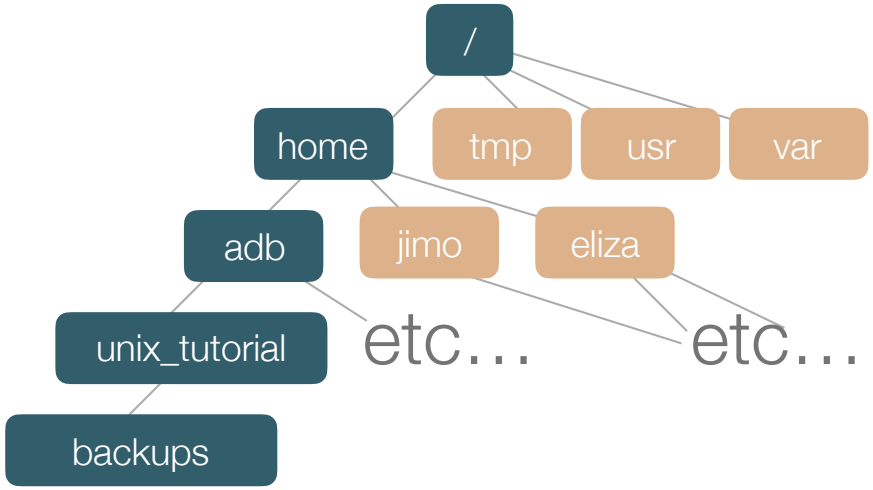
```
$ cd ..
```

- You can navigate up the hierarchy using successive ../../../

```
$ ls ../../
$ cd ../../
$ pwd
```

- or the full path up the hierarchy

```
$ cd /home/adb/unix_tutorial
$ pwd
$ ls
```



Hands-on: practice with cd to navigate the file system, ways to get home

If you are following along, you are in /home (can check with **`pwd`**, and when you **`ls`** you can see everyone else home directory
Note: If you ever need to get back to your home directory you can just type **`cd`** without any options:

```
$ cd
```

- the tilde is an alias to your home directory:

```
$ cd ~
```

- or you can specify the full path:

```
$ cd /home/adb
```

- or use the $HOME environment variable (on many systems:

```
$ cd $HOME
```

Summary: Very Basic Unix Commands

command	meaning
`ls [-a]`	list files and directories
`pwd`	display
`mkdir`	make a directory
`rmdir`	remove a(n empty) directory
`cd [directory]`	change to named directory
`cd ..`	change to parent directory
`cd ../[../../]`	change to successive parent directories up the hierarchy
`man [command]`	get help on a command
`cd` or `cd ~`	change to home directory
`ssh [system]`	securely login to a remote machine

Hands on: cp

- Go into you unix_tutorial directory:

```
$ cd ~/unix_tutorial
```

remind me to tell you about tab-completion!

- Copy a file from my directory to your directory:
  Usage `cp <file1> <file2>`

```
$ cp /home/adb/theyearofspaghetti.txt .
$ ls -lt
```

- Notice the dot! The dot indicates the current directory. We are copying a file to our current directory and not changing the name. If you wanted to change the name you could:

```
$ cp /home/adb/theyearofspaghetti.txt spaghetti.txt
```

Hands on: mv, rm

- Let's move the spaghetti.txt to our backups directory

```
$ mv spaghetti.txt backups
```

```
$ ls -lt
$ ls -lt bakcups
```

- the `mv` command is also used to rename files!

```
$ mv theyearofspaghetti.txt spaghetti.txt
```

- Create an extra copy of the text file so you can remove it:

```
$ cp theyearofspaghetti.txt tmpfile.txt
$ rm tmpfile.txt
```

Hands-on: basic file manipulation (cat, less, head)

- Clear the terminal window:

```
$ clear
```

- Display the contents of a file on a screen:

```
$ cat theyearofspaghetti.txt
```

- Display the contents of a file on a screen, a page at a time:

```
$ less theyearofspaghetti.txt
```

- Display the first 10 lines of a file to the screen:

```
$ head theyearofspaghetti.txt
$ tail theyearofspaghetti.txt
```

Hands-on: searching files (less, grep)

- You can search through a text file for a keyword (pattern) using `less`

```
$ less theyearofspaghetti.txt
```

then, still in `less`

```
$ /spaghetti
```

`less` finds and highlights the search term. Type n to go to the next occurrence of the word

`grep` searches for files for specified words or patterns:

```
$ clear
$ grep spaghetti theyearofspaghetti.txt
$ grep Spaghetti theyearofspaghetti.txt
```

`grep` is case sensitive use the `-i` option to make it case insensitive

```
$ grep -i spaghetti theyearofspaghetti.txt
```

Hands-on: searching files (grep, wc)

- search for a phrase, enclose in single quotes:

```
$ grep “cooking spaghetti” theyearofspaghetti.txt
```

- How many words/lines is in this story?

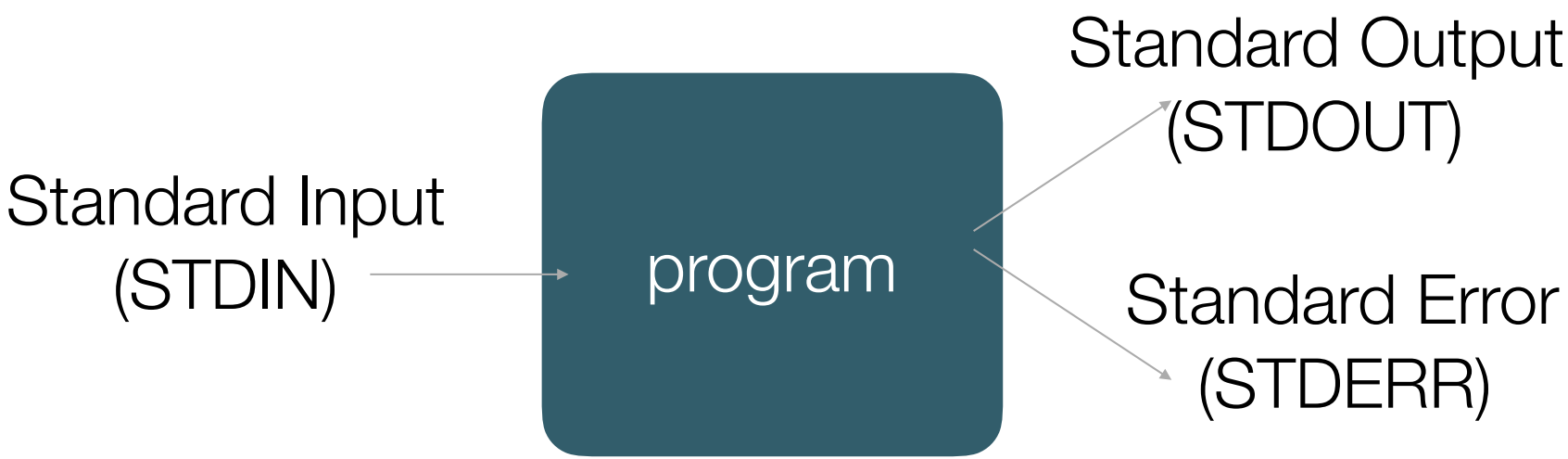
```
$ wc -w  theyearofspaghetti.txt
$ wc -l  theyearofspaghetti.txt
```

Summary: more Unix Commands

command	meaning
`cp <file1> <file2>`	copy file1 to file2
`mv <file1> <file2>`	move/rename file1 to file2
`rm`	remove a file
`cat`	display a file
`more/less <file>`	display a file one screen at a time
`head/tail <file>`	display first/last 10 lines of file
`grep 'word' <file>`	search a file for 'word'
`wc <file>`	count number of word/lines/chars

Redirecting output

- Typically in a UNIX environment you type in the name of a program and some command line options.
- The shell establishes 3 separate I/O streams: standard input, standard output and standard error.
- Most processes initiated by UNIX commands write to standard output (ie the screen), and take their input from standard input (the keyboard). There is also standard error.



Redirecting output

- The shell can attach things other than your keyboard to standard input:
    - A File (the contents of the file are fed to a program as if you typed it)
    - A pipe (the output of another program is fed as input as if you typed it)
- The shell can also attach things other than your screen to standard output:
    - A File (the output of a program is stored in a file)
    - A pipe (the output of another program is fed to the input of another program)

Hands-on: redirecting output

- To tell the shell to store the output of your program in a file, follow the command line for the program with the “>” character, followed by the filename:

```
$ ls /home > accounts.log
$ cat accounts.log
```

- You can concatenate two files using cat, and redirect the output to a new file:

```
$ cat theyearofspaghetti accounts.log > textblob.txt
```

- “>” will create (or overwrite) the file, if you want to append to and existing file use “>>”

```
$ whoami >> accounts.log
$ cat accounts.log
```

Hands-on: redirecting input

- the "**sort**" command will sort alphabetically or numerically, it takes input from standard input (the keyboard). To see how it works, type sort followed by a list of three fruits and then the key combo [Ctrl] and [d] to end the input (the Ctrl-D is the 'end of file' control sequence in UNIX

```
 $ sort
mango
tomato
apple
<Ctrl][d]>
```

- type "**who**" to see who is currently logged into the system, and redirect the output from the screen to a file

```
$ who
$ who > users.log
```

- redirect the input of users.log to the command sort

```
$ sort < users.log
```

- redirect the sorted output from standard out to a file

```
$ sort < users.log > sorted_users.log
```

Pipes

- A pipe is a holder for a stream of data
- A pipeline is a set of processes chained by their standard streams, so that the output of each process (stdout) feed directly as input (stdin) of the next one
- This is handy for using multiple commands together to perform a task

Hands-on: pipes

grep has a -o option which will output the word ONLY

- How many times does Haruki Muramaki use the word spaghetti in this short story?

```
$ grep -oi “spaghetti” theyearofspaghetti.txt | wc-l
```

- other examples (| `more` useful in many cases when you have more than a screenful of text, | `grep` useful when you are looking for something) :

```
$ history | grep grep
$ ls -lat | more
$ who | sort
```

Summary: redirection and other Unix Commands

command	meaning
`command > file`	redirect stdout to file
`command >> file`	append stdout to file
`command < file`	redirect stdin from a file
`command1 | command2`	pipe the stdout of command1 to the stdin of command2
`cat file1 file2 > file3`	concatenate file1 and file2
`sort`	sort data
`who`	list of current users logged in
`wc <file>`	count number of word/lines/chars

Hands on: Introduction to UNIX/LINUX

1. Very Basic Commands/Interacting with the shell
2. **File attributes and permissions**
3. Job control
4. UNIX/Linux Environment variables
5. Text Editors
6. Remote Access

File Attributes and Permissions

```
[adb@isp02 ~]$ ls -l
total 20
-rw-r----- 1 adb TACC-VIS-SVT    0 Aug 29 11:21 answer_key.txt
-rwxrwxrwx 1 adb G-25503        35 Aug 29 11:19 logbook.txt
-r-------- 1 adb G-25503        20 Aug 29 11:17 seekrets.txt
-rw-r--r-- 1 adb G-25503      9135 Aug 29 10:25 theyearofspaghetti.txt
drwxr-xr-x 3 adb G-25503        20 Aug 29 09:48 unix_tutorial
```

permissions | owner | group | size | last modified time/date | file/directory name

number of links or directories inside the directory

time/size attributes	
command	meaning
`ls -l`	when the file was last modified
`ls -lc`	when the file was last changed (change of wonder, permissions, etc)
`ls -lu`	when the file was last accessed
`ls -lt`	chronological listing
`ls -lh`	“human readable” size
`stat <filename>`	display date-related attributes

File Attributes and Permissions

```
-rw-r----- 1 adb TACC-VIS-SVT
lrwxrwxrwx 1 adb G-25503
-rwxrwxrwx 1 adb G-25503
-r-------- 1 adb G-25503
-rw-r--r-- 1 adb G-25503      9
drwxr-xr-x 3 adb G-25503
```

permissions owner group

- Each file has a set of *permissions* that control who can can access the file (and how). There are three different types of permissions:
  - read `r`, write `w`, execute `x`
  - In Unix/Linux, there are permission levels associated with three types of people that might access a file:
    - owner (you)
    - group (a group of other users that you set up or belong to)
    - world (anyone else browsing around on the file system)

first column specifies type:

`-rwxrwxrwx`

Owner Group Others

Symbol	File type
-	plain file
d	directory
l	symbolic link

File Attributes and Permissions

```
-rw-r------ 1 adb TACC-VIS-SVT
lrwxrwxrwx 1 adb G-25503
-rwxrwxrwx 1 adb G-25503
-r-------- 1 adb G-25503
-rw-r--r-- 1 adb G-25503      9
drwxr-xr-x 3 adb G-25503
```

permissions owner group

- Meaning for Files:
  - `r` allowed to read
  - `w` allowed to write
  - `x` allowed to execute
- Meaning for Directories:
  - `r` allowed to see names of the files
  - `w` allowed to add and remove files
  - `x` allowed to enter the directory

-rwxrwxrwx

Owner Group Others

Symbol	File type
-	plain file
d	directory
l	symbolic link

Hands-on: File Attributes and Permissions

```
$ cd /home
$ ls
```

you should see all the home directories on the system.

- View the directory contents with “long format” to get more information and chronologically (sort by time with the newest first)

```
$ ls -lt
```

Hands-on: File Attributes and Permissions

- Notice my directory “adb” and the permissions. I have allowed group/world access to enter the directory (`x`) and see names of the files (`r`)

```
drwxr-xr-x    5 adb       G-25503          4096 Aug 29 14:16 adb
drwxr-xr-x. 187 root      root             8192 Aug 29 09:19 .
drwx------   12 nlemcke   G-816966        4096 Aug 25 11:43 nlemcke
drwx------   26 jag7548   G-816966        4096 Aug 24 17:12 jag7548
drwxr-xr-x   26 charlie   G-80748         4096 Aug 24 13:55 charlie
drwx------   13 kpereida  G-816966        4096 Aug 12 22:59 kpereida
```

- The directory listing might have ran off the page, to view the text one screen full at a time:

```
$ ls -lt | more
```

Hands-on: File Attributes and Permissions

- Enter the adb directory and list the contents in human-readable format:

```
$cd adb
$ls -lht
```

```
-rw-r----- 1 adb TACC-VIS-SVT    0 Aug 29 11:21 answer_key.txt
lrwxrwxrwx 1 adb G-25503        21 Aug 29 14:16 backups -> unix_tutorial/backups
-rwxrwxrwx 1 adb G-25503        35 Aug 29 11:19 logbook.txt
-r-------- 1 adb G-25503        20 Aug 29 11:17 seekrets.txt
-rw-r--r-- 1 adb G-25503      9.0K Aug 29 10:25 theyearofspaghetti.txt
drwxr-xr-x 3 adb G-25503        20 Aug 29 09:48 unix_tutorial
```

- Next we will modify permissions in our own home directories to see how to set up a shared folder as well as a private folder.

Changing File Permissions

- The chmod command changed the permissions associated with the file or directory
- Syntax **`chmod [mode] <filename>`**
- The mode can be specified by a symbolic representation or an octal number
- Both methods achieve the same result
- Multiple symbolic operations can be given separated by commas.

```
$cd adb
$ls -lht
```

```
-rw-r----- 1 adb TACC-VIS-SVT    0 Aug 29 11:21 answer_key.txt
lrwxrwxrwx 1 adb G-25503        21 Aug 29 14:16 backups -> unix_tutorial/backups
-rwxrwxrwx 1 adb G-25503        35 Aug 29 11:19 logbook.txt
-r-------- 1 adb G-25503        20 Aug 29 11:17 seekrets.txt
-rw-r--r-- 1 adb G-25503      9.0K Aug 29 10:25 theyearofspaghetti.txt
drwxr-xr-x 3 adb G-25503        20 Aug 29 09:48 unix_tutorial
```

chmod: symbolic representation

- TODO

chmod: Symbolic Representation

- *Symbolic* Mode representation has the following form:

[ugoa] [+-=] [rwxX…]

u=user	**+** add permission	**r**=read
g=group	**-** remove permission	**w**=write
o=other	**=** set permission	**x**=execute
a= all		**X**= see below

- The **X** permission option is very handy - it sets to execute only if the target is a directory or already has execute permission

Hands-on: chmod, symbolic representation

- TODO

chmod: octal representation

- TODO

chmod: Octal Representation

- Octal Mode uses a single argument string which describes the permissions for a file (3 digits)
- Each digit of this number is a code for each of the three permission levels (user, group, world)
- Permissions are set according to the following numbers:
  - Read = 4
  - Write = 2
  - Execute = 1

	Permission Level
0	no permissions
1	execute only
2	write only
3	write and execute (1+2)
4	read only
5	read and execute (4+1)
6	read and write (4+2)
7	read, write and execute (4+2+1)

- Sum the individual permissions to get the desired combination

Hands-on: chmod, octal representation

- TODO

Hands on: Introduction to UNIX/LINUX

1. Very Basic Commands/Interacting with the shell
2. File attributes and permissions
3. **Job control**
4. UNIX/Linux Environment variables
5. Text Editors
6. Remote Access

Job Control

- TODO

Job Control

- The shell allows you to manage jobs
  - place jobs in the background
  - move a job to the foreground
  - suspend or kill a job
- If you follow a command line with "**&**", the shell will run the job in the background
  - you don't need to wait for the job to complete
  - you can type in a new command right away
  - you can have a bunch of jobs running at once

Job Control

- TODO: Hands on Exercises

Hands on: Introduction to UNIX/LINUX

1. Very Basic Commands/Interacting with the shell
2. File attributes and permissions
3. Job control
4. **UNIX/Linux Environment variables**
5. Text Editors
6. Remote Access

Environment Variables

- TODO (most hands-on)
- echo
- env
- env | grep

Customizing the Shell

- TODO

Customizing the Shell

- Each shell supports some customization.
  - user prompt settings
  - environment variable settings
  - aliases
- The customization takes place in startup files which are read by the shell when it starts up
  - Global files are read first - these are provided by the system administrators (e.g.. /etc/profile)
  - Local files are then read in the user's HOME directory to allow for additional customization

Hands on: Customizing the Shell

- TODO

Shell Startup Files

sh,ksh:

```
~/.profile
```

bash:

```
~/.bash_profile
~/.bash_login
    ~/.profile
    ~/.bashrc
    ~/.bash_logout
```

csh:

```
~/.cshrc
~/.login
~/.logout
```

tcsh:

```
~/.tshrc
~/.cshrc
~/.login
~/.logout
```

Note: on TACC production systems, we provide an alternative location for customization files to avoid overriding system defaults:

BASH:	`~/.profile_user`
CSH/TCSH:	`~/.login_user`
	`~/.cshrc_user`

Hands on: Introduction to UNIX/LINUX

1. Very Basic Commands/Interacting with the shell
2. File attributes and permissions
3. Job control
4. UNIX/Linux Environment variables
5. **Text Editors**
6. Remote Access

Text Editors: vi, emacs, pico, nano

Hands on: Text Editors: vi, emacs, pico, nano

Text Editors: vi, emacs, pico, nano

Hands on: Introduction to UNIX/LINUX

1. Very Basic Commands/Interacting with the shell
2. File attributes and permissions
3. Job control
4. UNIX/Linux Environment variables
5. Text Editors
6. **Remote Access**

Remote Access and using TACC systems

- TODO, will be a separate slide deck