

# Introduction to Unix/Linux

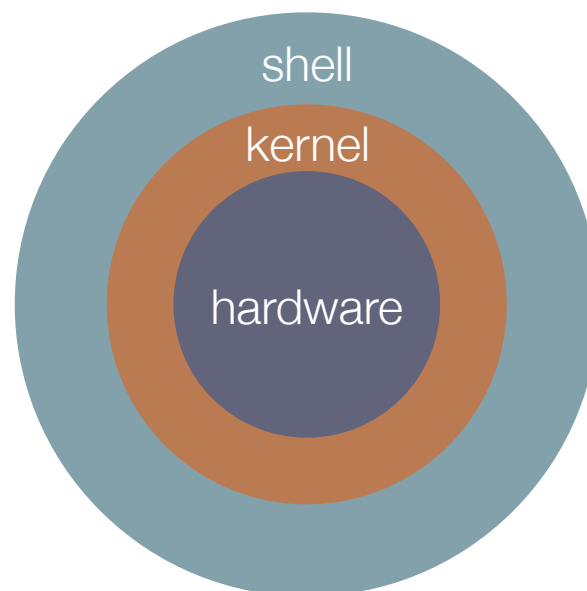
---

- Sept 7,12
- 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 several different 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](http://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. UNIX/Linux Environment variables
4. Job control
5. Text Editors

# Hands-on: pwd, ls

---

- When you first login, your 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: using ls \*, find, locate

---

- List all of the shared library files (.so files in /usr/lib)

```
$ ls /usr/lib/*.so
```

- Or all the text files (.txt) in your local directory:

```
$ ls -lat /home/adb/*.txt
```

Where did I put that text file?

**find** lets you find files with various attributes on your filesystem  
(more later, useful tutorial <http://www.grymoire.com/Unix/Find.html>)

```
$ find /home/adb/ -iname *list*
```

Where are the GL header files on the system?

**locate** will find all instances of file

(more later, useful tutorial <https://www.unixtutorial.org/commands/locate/> )

```
$locate GL
```

# 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:

```
$ 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 ../..
```

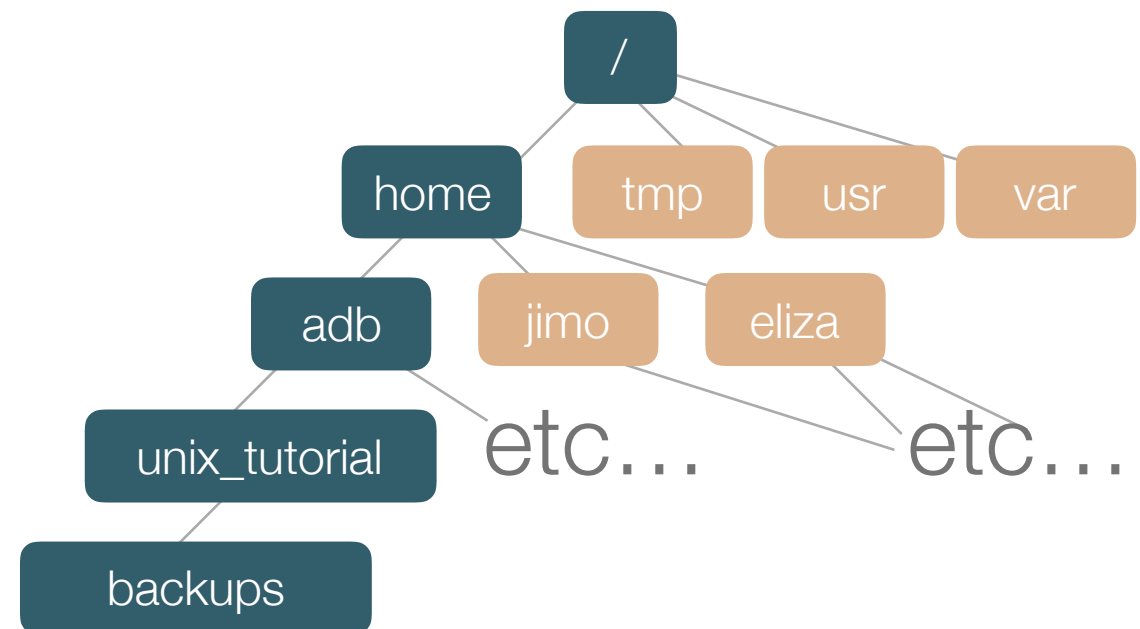
```
$ pwd
```

- 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 a shortcut 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
<code>ls [-a]</code>	list files and directories
<code>pwd</code>	display
<code>mkdir</code>	make a directory
<code>rmdir</code>	remove a(n empty) directory
<code>cd [directory]</code>	change to named directory
<code>cd ..</code>	change to parent directory
<code>cd .. [../..]</code>	change to successive parent directories up the hierarchy
<code>man [command]</code>	get help on a command
<code>cd</code> or <code>cd ~</code>	change to home directory
<code>ssh [system]</code>	securely login to a remote machine
<code>find, locate</code>	useful for finding files

# Hands on: cp

---

- Go into you unix\_tutorial directory:

```
$ cd ~/unix_tutorial
```

- Copy a file from my directory to your directory:

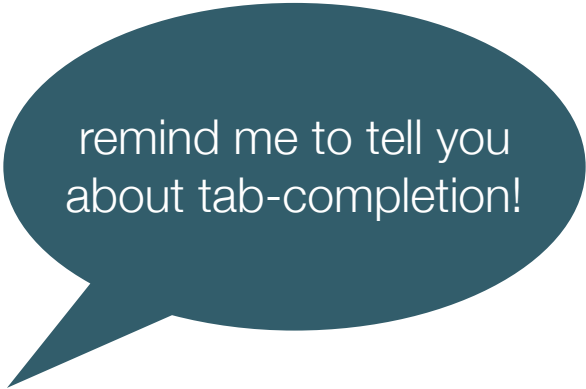
Usage `cp <file1> <file2>`

```
$ 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
```



remind me to tell you  
about tab-completion!

# Hands on: mv, rm

---

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

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

```
$ ls -lt
```

```
$ ls -lt backups
```

- 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
```

- Be very careful with rm \*

If you are worried you can alias rm to rm -i or rm -I so the OS will prompt you before removing a file (we will see how to create aliases later), see man rm

# 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`, search with /

```
/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
```

- grep has many useful options `man grep` and google for resources (<https://www.computerhope.com/unix/ugrep.htm>, for example) we will see more sophisticated examples later (parsing data with grep awk and sed)

# Summary: more Unix Commands

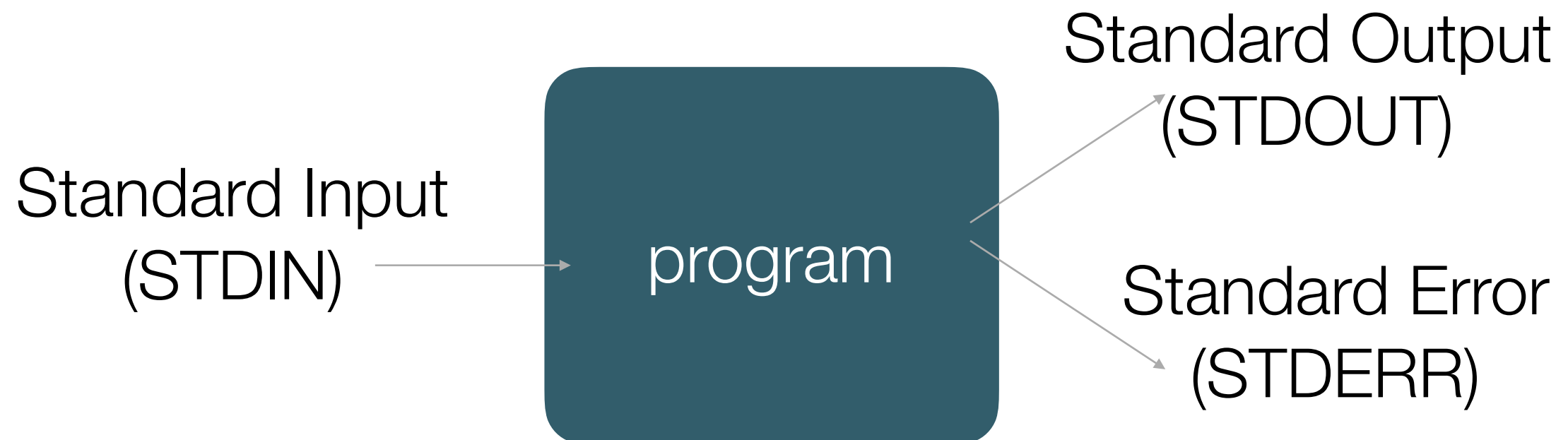
---

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

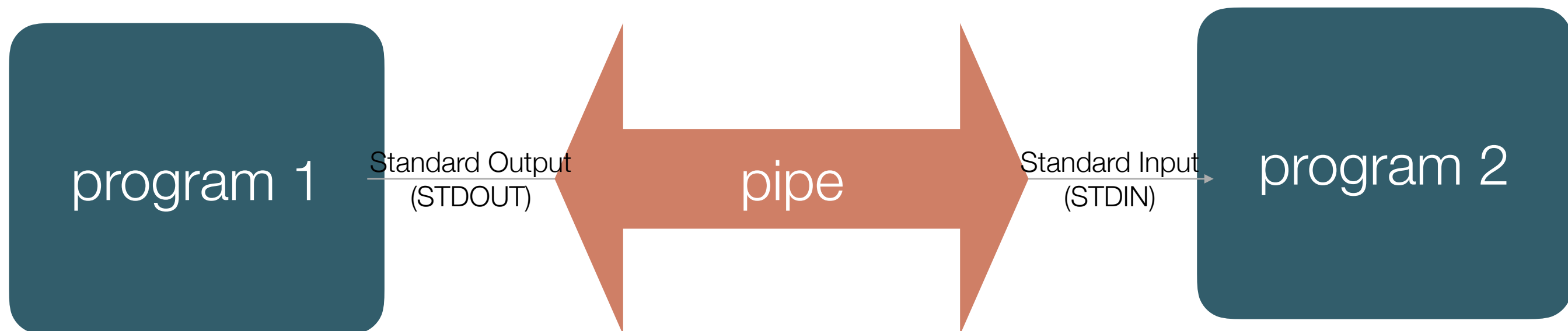
- redirect the sorted output from standard out to a file

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

# Pipes

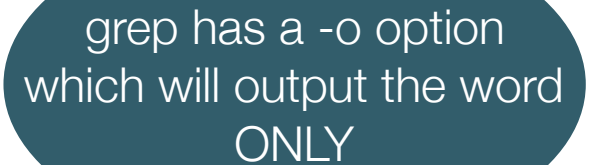
---

- 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
```

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

- other examples ( | less 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 | less
```

```
$ who | sort
```

# sed, awk and more advanced scripting preview

---

- We will learn about some more advanced file parsing with sed/awk later....but just to give you an idea:

```
7559640
crov-merge
C      -796.608      -44.387      1252.402
C      -796.461      -41.003      1253.900
C      -797.352      -38.541      1251.234
• etc
```

We will cover awk/sed more later in the data portion of the course...but here are some good tutorials if you can't wait:

<http://www.grymoire.com/Unix/Sed.html>

<http://www.grymoire.com/Unix/Awk.html>

- Example: malformed huge xyz file for an extremely large "girus"
  - needed to reorder the columns and substitute the atomic number for each element
  - very large file

```
grep '^[HNCOS]' crov-allatoms.xyz | awk '{print $2,$3,$4,$1}' | sed 's/N$/7/' | sed 's/C$/6/' | sed 's/S$/17/' | sed 's/H$/1/' | sed 's/O$/8/' > crov-allatoms.txt
```

- The ^ anchor specifies that the pattern following it should be at the start of the line
  - > awk which will print out columns, 2,3,4 and then 1
  - > sed will search for a N at the end of the line (\$), and replace it with a 7 etc

# Summary: redirection and other Unix Commands

---

command	meaning
<code>command &gt; file</code>	redirect stdout to file
<code>command &gt;&gt; file</code>	append stdout to file
<code>command &lt; file</code>	redirect stdin from a file
<code>command1   command2</code>	pipe the stdout of command1 to the stdin of command2
<code>cat file1 file2 &gt; file3</code>	concatenate file1 and file2
<code>sort</code>	sort data
<code>who</code>	list of current users logged in
<code>wc &lt;file&gt;</code>	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. UNIX/Linux Environment variables

4. Job control

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 owner, 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 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:

-	<b>rwxrwxrwx</b>		
	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	

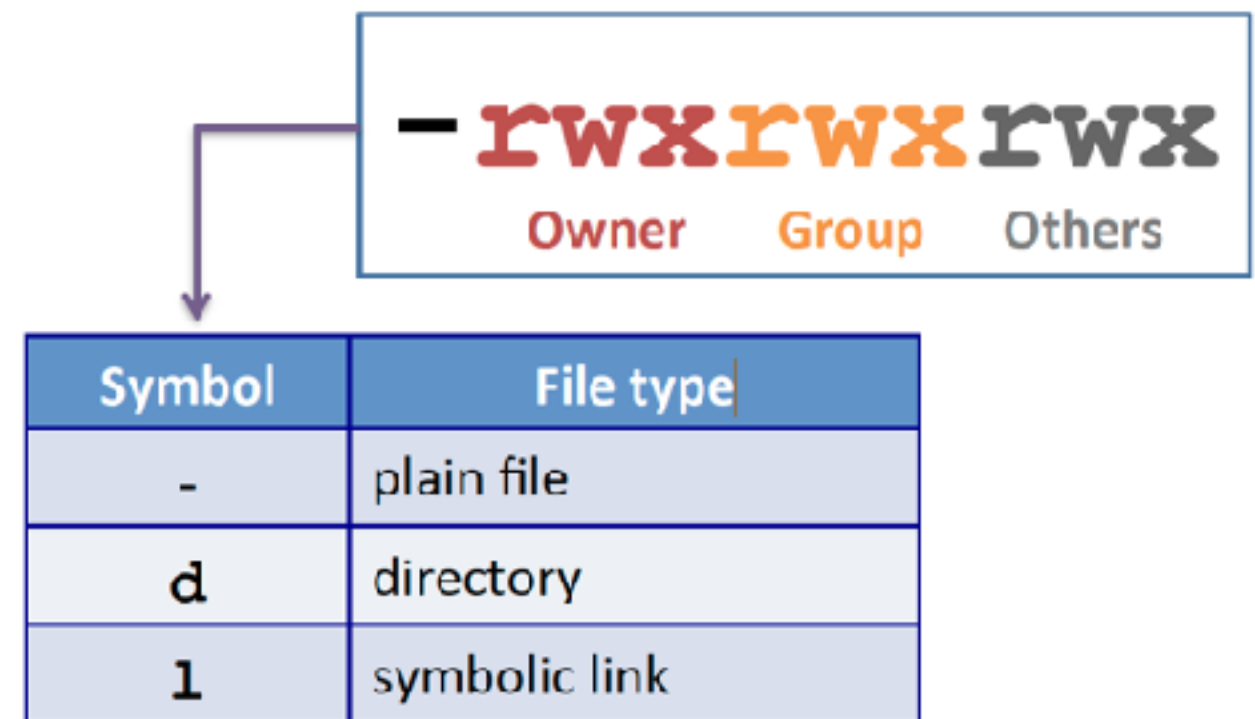
permissions owner group

- 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 (This implies that you may read files in the directory provided you have read permission on the individual files.)



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 | less
```

# 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. Only the owner of a file can use `chmod` to change the permissions of a file.
- 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

---

- 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 execute*
*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

---

Let's experiment with changing permissions with files in our home directory:

- Go to your `unix_tutorial` folder in your home directory, note that the group for the class is G-819394:

```
$ cd ~/unix_tutorial  
$ ls -lat
```

- Experiment with changing the owner (to yourself), group and permissions of `theyearofspaghetti.txt`

```
$ chgrp G-819394 theyearofspaghetti.txt  
$ chown dmcdougall theyearofspaghetti.txt  
$ chmod u-w,g+x,o=x theyearofspaghetti.txt
```

- Questions?

# 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 bit will add 4 to the total (in binary 100)  
Write bit will add 2 to the total (in binary 010)  
Execute bit will add 1 to the total (in binary 001)  
Sum the individual permissions to get the desired combination. (e.g. if you want the owner only to have rwx permissions, group to have read only and world to have no access:

	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)

owner	• 4(r)+2(w)+1(e)	7
group	• 4(r)+0(w)+0(e)	4
world	• 0(r)+0(w)+0(e)	0
<b>chmod 740 &lt;filename&gt;</b>		

# Hands-on: chmod octal representation practice

---

think about it before you try it  
and see the solution

- What would the following do:

```
$chmod 777 theyearofspaghetti.txt
```

```
$chmod 640 theyearofspaghetti.txt
```

```
$chmod 744 theyearofspaghetti.txt
```

Read = 4

Write = 2

Execute = 1

Sum the individual permissions to get the desired combination.

	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)

# Hands-on Exercise: permissions for a shared and private folder (symbolic or octal, your choice)

---

- Make a directory called “shared” and a directory called “private” and observe the permissions (ls -lat):

```
[adb@isp02 unix_tutorial]$ mkdir shared private
[adb@isp02 unix_tutorial]$ ls -lat
total 16
drwxr-xr-x 5 adb G-25503 76 Sep 1 12:20 .
drwxr-xr-x 2 adb G-25503 6 Sep 1 12:20 private
drwxr-xr-x 2 adb G-25503 6 Sep 1 12:20 shared
```

We want to make the permissions of the shared directory and the files inside open (rwx) for anyone in the group G-819394 and for the owner. We want to let anyone else on the system (ie world) be able to read the files only. To do that we need to make the parent directories readable and executable (**drwxr-xr-x**), and all the files inside readable, writable to the group, and just readable to the world (**-rwxrw-r-**). Note\* since this is just a text file, we don't need to worry about the executable permissions on the file.

Also, we want to make a private folder that is readable and writable only by the owner (you) (**drwx-----**)

How would you do this? Give it a shot and then partner up with someone to see if they can access your directory and file (we won't worry about writing to the file just yet)



# Hands on: Introduction to UNIX/LINUX

---

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

# 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.

# 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

# Useful bash Aliases

---

- prevent common typo: `alias sl="ls"`
- prevent common typo protect yourself from `rm *`: `alias rm="rm -i"`
- list all files ordered by date: `alias lt="ls -alrt --color=auto"`
- use arguments: `alias histg="history | grep"`
- include burger in prompt: `export PS1="\u@\h \w 🍔 $ "`
  - `\u` username
  - `\h` hostname
  - `\w` current directory
- find more: UNICDOE list of emoji's  
<https://emojipedia.org/unicode-8.0/>
- google "unix prompt options" to get UNICODE for many symbols

# Shell start-up 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 alternate location for customization files to avoid overriding system defaults:

BASH: ~/.profile\_user  
CSH/TCSH: ~/.login\_user  
          ~/.cshrc\_user

# Environment Variables:

---

- Unix/Linux shells maintain a list of environment variables which have a unique name and a value associated with them:
  - some of these parameters determine the behavior of the shell
  - also determine which programs get run when commands are entered (and which libraries they link against)
  - provide information about the execution environment to programs
- We can access these variables:
  - set new values to customize the shell
  - find out their values to accomplish a task

# Hands-on: Environment Variables

---

- To view environment variables, use the `env` command

```
$ clear  
$ env
```

- If you know what you are looking for, you can use `grep`

```
$ env | grep PWD  
PWD=/home/adb
```

- Use the `echo` command to print variables the `$` prefix is required to access the value of the variable:

```
$ echo $PWD  
/home/adb
```

- Can also use the environment variable in conjunction with other commands:

```
$ ls $PWD
```

# Special Environment Variable: PATH

---

- Each time you provide the shell a command to execute, it does the following:
  - Checks to see if the command is a built-in shell command
  - If it is not a built-in command, the shell tries to find a program whose name matches the desired command
- How does the shell know where to look on the file system  
The PATH variable tells the shell where to search for programs (non built in commands)



# Special Environment Variable: PATH

---

- See what is in your \$PATH

```
[adb@isp02 ~]$ echo $PATH
/opt/intel/advisor_2017.1.1.486553/bin64:/opt/intel/vtune_amplifier_xe_2017.1.0.486011/bin64:/opt/intel/inspector_2017.1.1.484836/bin64:/opt/intel/itac/2017.1.024/intel64/bin:/opt/intel//itac/2017.1.024/intel64/bin:/opt/intel//itac/2017.1.024/intel64/bin:/opt/intel//clck/2017.1.016/bin/intel64:/opt/intel/compilers_and_libraries_2017.1.132/linux/bin/intel64:/opt/intel/compilers_and_libraries_2017.1.132/linux/mpi/intel64/bin:/opt/intel/debugger_2017/gdb/intel64_mic/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/intel/bin:/usr/bin/tester:/home/adb/.local/bin:/home/adb/bin
```

- You can add more search directories to you PATH by changing the shell startup files:
  - BASH: export PATH=\$PATH:/home/user/bin
  - TCSH: set path = (/home/user/bin \$PATH)

-

# Other important variables:

---

PWD	current working directory
MANPATH	where to find man pages
HOME	home directory of user
MAIL	where your e-mail is stored
TERM	what kinds of terminal you have
PRINTER	specifies the default printer name
EDITOR	used by many apps to identify editor pref (vi, emacs)
LD_LIBRARY_PATH	search path for dynamic runtime library

# Setting Environment Variables:

---

- The syntax depends on your shell (for bash use export, tcsh use setenv):

```
$ export FRUIT=mango  
$ echo $FRUIT  
mango
```

- Note: environment variables that you set interactively are only available in your current shell:
  - if you spawn a new shell (e.g. `xterm&` or open a new tab via a GUI) these settings will be lost
  - to make permanent changes you can alter the login scripts that affect your shell  
e.g. `.profile_user`
- (more on environment variables and the TACC module system coming soon)

# Hands-on: Customizing your .bashrc

---

- On most production TACC systems the changes to your profile go in .profile\_user but on [isp.tacc.utexas.edu](http://isp.tacc.utexas.edu), we can edit our .bashrc

```
$ vi ~/.bashrc
```

- For practice, try creating an alias (as shown in [previous slide](#)) or change your prompt (<http://www.linuxnix.com/linuxunix-shell-ps1-prompt-explained-in-detail/>) :

- Check to see if it works

```
$ source ~/.bashrc
```

# Hands on: Introduction to UNIX/LINUX

---

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

# 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 a “&”, 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

# Hands-on: 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. UNIX/Linux Environment variables
4. Job control
- 5. Text Editors**
6. Remote Access



# Text Editors: vi, emacs, nano

---

- From Wikipedia: "vi is a modal editor: it operates in either *insert mode* (where typed text becomes part of the document) or *normal mode* (where keystrokes are interpreted as commands that control the edit session). For example, typing i while in normal mode switches the editor to insert mode, but typing i again at this point places an "i" character in the document. From insert mode, pressing ESC switches the editor back to normal mode." It can be found on nearly every UNIX/Linux system in existence. To exit, type :q
- Emacs is not modal but has thousands of commands and macros, usually in the form of a 1 or 2-key sequence (e.g. CTRL-n: "go to end of line"). To exit, type CTRL-X CTRL-C.
- Nano are more straightforward and show their common commands at the bottom. To exit, type CTRL-X

# Hands on: Text Editors: vi, emacs, nano and escape characters

---

- search and replace in vi

```
$ vi theyearofspaghetti.txt  
:%s/Id/I\'d/
```

- Notice we had to escape the ' using the back slash. The backslash (\) character is used to mark these special characters so that they are not interpreted by the shell, but passed on to the command being run (for example, **echo**).
- Here is a list of characters that have special meaning to the shell and might need to be escaped <https://unix.stackexchange.com/questions/270977/what-characters-are-required-to-be-escaped-in-command-line-arguments/270979>
- We will revisit this topic when get to some more advanced scripting topics (here is a good tutorial <https://www.shellscript.sh/escape.html>)

# 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