Introduction to the Linux Environment

Victor Eijkhout

2014/10/09

Linux in the Real (supercomputing) World

- Q: Before we begin, does anyone have a feel for how many machines in the November 2013 Top500 list ran variants of the Linux operating System?

- Q: How about Windows?

Linux in the Real World

A: 97% are Linux-like

Operating System	# of Systems	Percentage
Linux	462	92.4%
Unix	22	4.8%
Mixed	11	2.2%
Windows	2	0.4%
BSD Based	1	0.20%

Unix
A Little History

- Unix is over 40 years old!
- Unix dates back to 1969 with a group at Bell Laboratories
- The original Unix operating system was written in assembler
- First 1972 Unix installations had 3 users and a 500KB disk

DEC PDP-11, 1972

Linux
Bringing Unix to the Desktop

- Unix was very expensive
- Microsoft DOS was the mainstream OS
- MINIX, tried but was not a full port
- An open source solution was needed!

1990's Movers and Shakers

Richard Stallman, father of the GNU Project

Linus Torvalds

What is Linux?

- Linux is a clone of the Unix operating system written from scratch by Linus Torvalds with assistance from developers around the globe (technically speaking, Linux is not Unix)
- Torvalds uploaded the first version - 0.01 in September 1991
- Only about 2% of the current Linux kernel is written by Torvalds himself but he remains the ultimate authority on what new code is incorporated into the Linux kernel.
- Developed under the GNU General Public License , the source code for Linux is freely available
- A large number of Linux-based distributions exist (for free or purchase)

Gnu-Linux

- The real Linux stuff is actually deep down: process and file management
- Normally you talk to the “shell”: command for the user to access that file & process functionality.
- Different shells: sh/bash/csh/tcsh/zsh/ksh
  Many similarities; where they differ I will teach “bash”
- Also different Linux distributions, but the differences are deeper down and less important.

Why use Linux?

- Performance: as we've seen, supercomputers generally run Linux; rich-multi user environment
- Functionality: a number of community driven scientific applications and libraries are developed under Linux (molecular dynamics, linear algebra, fast-fourier transforms, etc).
- Flexibility/Portability: Linux lets you build your own applications and there is a wide array of support tools (compilers, scientific libraries, debuggers, network monitoring, etc.)

Why Linux is Still Used

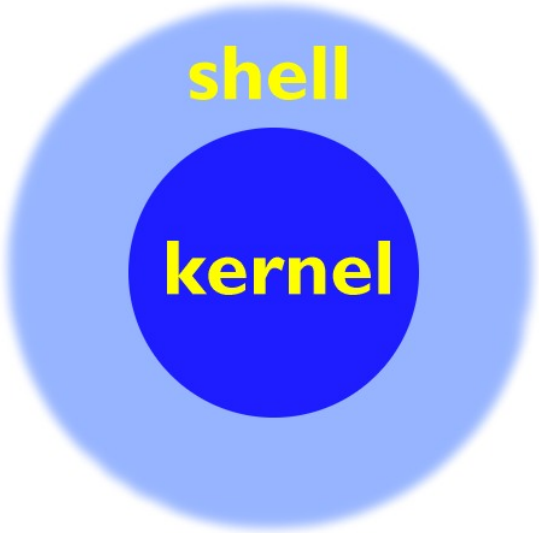
- 40+ years of development (Unix)
  - Linux 1991
- Many academic, scientific, and system tools
- Open Source
- System Stability
- Lightweight
- Easy Development

The Basics

- The Command Line
  - Interaction with Linux is based on entering commands to a text terminal
  - Often there are no 'warnings' with commands, no 'undo'
- The Shell
  - The user environment that enables interaction with the kernel, or lower-system OS.
  - Windows Explorer would be a shell for Microsoft Windows.

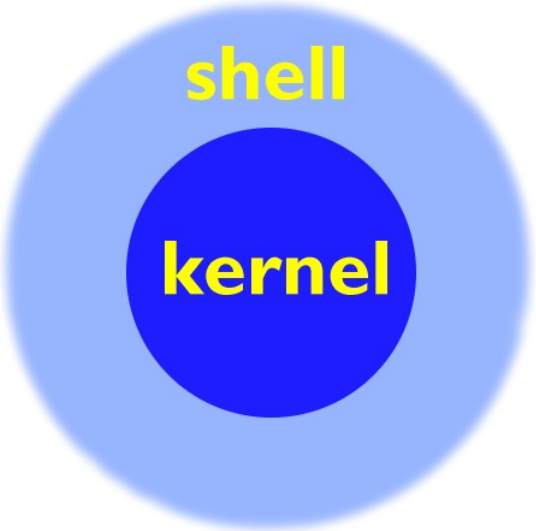
The Basics
How does Linux work?

- Linux has a kernel and one or more shells
- The kernel is the core of the OS; it receives tasks from the shell and performs them
- The shell is the interface with which the user interacts



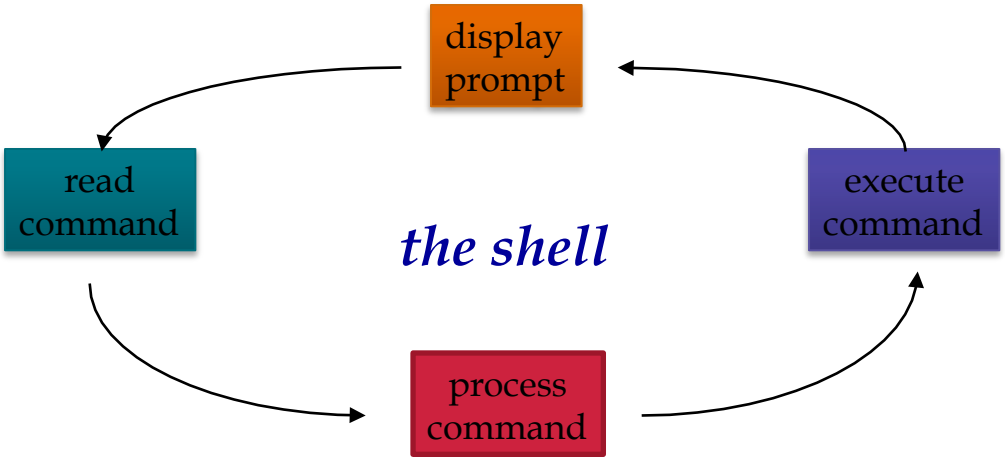
The Basics
How does Linux work?

- Everything in Linux is either a file or a process
- A process is an executing program identified by a unique PID (process identifier). Processes may be short in duration or run indefinitely
- A file is a collection of data. Files are created by users using text editors, running compilers, etc
- The Linux kernel is responsible for organizing processes and interacting with files: it allocates time and memory to each processes and handles the filesystem and communications in response to system calls



The Basics
What does the Shell Do?

- The user interface is called the *shell*.
- The shell tends to do 4 jobs repeatedly:



The Basics
Linux Interaction

- The user interacts with Linux via a shell
- The shell can be graphical (X-Windows) or text-based (command-line) shells like tcsh and bash
- To remotely access a shell session on TACC production resources, use ssh (secure shell)

Open A Terminal

- Linux: use a window manager (fvwm, Gnome, &c)
- MacOSX – use X11 or the terminal program (Applications>Utilities)
- Windows – use Cygwin or VirtualBox, or download puTTY or SSH Client to use a remote machine.

Linux Accounts

- To access a Linux system you need to have an *account*
- Linux account includes:
  - username and password
  - userid and groupid
  - home directory
    - a place to keep all your snazzy files
    - may be quota'd, meaning that the system imposes a limit on how much data you can have
  - a default shell preference
- If you do this training on your laptop, all this is automatic.

Exercise 1

```
$ whoami
$ who
# suppose your name is "fred":
$ finger fred
```

These are a few commands relating to users on the system.

Files and File Names

- A file is a basic unit of storage (usually storage on a disk)
- Every file has a name
- File names can contain any characters (although some make it difficult to access the file)
- Unix file names can be long (how long depends on your specific flavor of Unix)
- File names starting with period are “hidden files”

Exercise 2

```
$ ls
$ ls -a
$ touch this-is-a-new+name
$ ls
# now read the "man page":
$ man ls
# (and type "q" to get out)
```

Listing of files, and creating an empty file.

Exercise: how do you find out the size of a file and the time it was last touched?

File Contents

- Each file can hold some raw data
- Linux does not impose any structure on files
  - files can hold any sequence of bytes
  - it is up to the application or user to interpret the files correctly
- Many programs interpret the contents of a file as having some special structure
  - text file, sequence of integers, database records, etc.
  - in scientific computing, we often use binary files for efficiency in storage and data access
    - Fortran unformatted files
    - Scientific data formats like NetCDF or HDF have specific formats and provide APIs for reading and writing

Exercise 3

```
$ cat > a_text_file
Just input some
text, doesn't matter what
^D
$ cat a_text_file
$ file a_text_file
$ wc a_text_file
```

Create a text file and display it

Show that it is a text file, and display some statistics on it

More about File Names

- Every file must have a name
- Each file in the same directory must have a unique name
- Files that are in different directories can have the same name
- Note: **Linux is case-sensitive**
  - So, “texas-fight” is different than “Texas-Fight”
    - **Mac caveat: MacOS is NOT cAsE sEnSiTiVe**

Directories

- A directory is a special kind of file - Unix uses a directory to hold information about other files
- We often think of a directory as a container that holds other files (or directories)
- Mac and Windows users can relate a *directory* to the same idea as a *folder*

Exercise 4

```
$ mkdir new_directory
$ ls -l
$ ls new_directory
```

Make a new directory,
recognize that it is a directory
see that it is empty

Directories

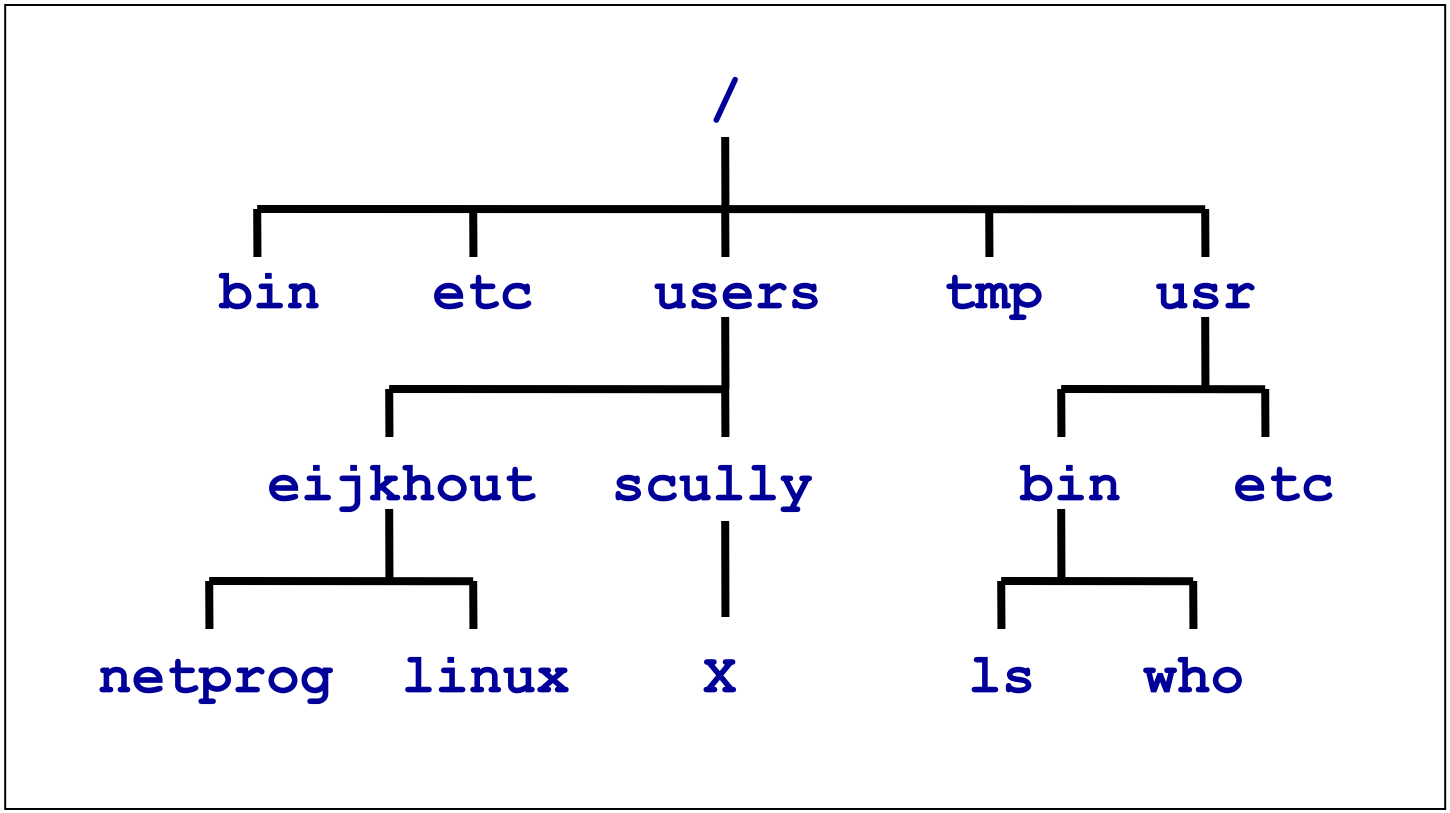
What is a *working directory*?

The directory your shell is currently associated with. At anytime in the system your login is associated with a directory

Directories form a tree, and you can go into subtrees or back out of them.

It is always easy to return to your *home directory*

Linux File System (an upside-down tree)



Exercise 5

```
$ touch not_directory
$ ls
$ ls -l
$ ls -F
$ ls *
$ ls -d *
```

Various ways of looking at files and directories

Exercise 6

```
$ pwd
$ cd new_directory
$ pwd
$ mkdir inner_directory
$ cd inner_directory
$ pwd
$ cd ..
$ pwd
$ cd
$ pwd
```

You made a new directory before, now make that your working directory with the “cd” (change directory) command.

Repeat…

The parent directory is always “..” change directory to the parent

It’s always easy to go back home

Relative vs.. Absolute Path

Commands expect you to give them a path to a file. Most commands will let you provide a file with a relative path, or a path relative to your working directory.

../directory - the '..' refers to looking at our previous directory first

./executable - '.' says this directory, or our working directory

Absolute, or Full paths are complete. An easy way to know if a path is absolute is does it contain the '/' character at the beginning?

/home/user/directory/executable - a full path to file executable

Exercise 7

```
$ cd # go home
$ pwd
$ cd /home/yourname/new_directory
$ cd ..
```

You made a new directory before, now make that your working directory with the “cd” (change directory) command.

Repeat…

The parent directory is always “..” change directory to the parent

Finding your home

Each user has a home directory which can be found with:

cd

cd *~eijkhout*

cd *$HOME*

The tilde character '~' will tell the shell to auto-complete the path statement for the **cd** command

$HOME refers to an *environment variable* which contains the path for home.

More file commands

cd *directory* - change your current working directory to the new path

ls *–a* – show hidden files

Hidden files are files that begin with a period in the filename '.'

mv - moves one file to another

cp – copies files or directories

rm – remove files & directories

rm *–rf* – remove everything with no warnings

rm *–rf* * - most dangerous command you can run!

rename from to filenames – can rename lots of files at once

- rename file file0 file?.txt (i.e. would move file1.txt to file01.txt)

Exercise 8

```
$ cd ~/new_directory
$ cat > test1
Some text
^D
$ cp test1 test2
$ ls
$ cat test2
$ mv test1 test2 inner_directory
$ ls
$ ls inner_directory
$
```

Play around with the “cp” and “mv” commands. They do different things with files and directories.

Recursive Directories

Oftentimes a manual will refer to 'recursive' actions on directories. This means to perform an action on the given directory and recursively to all subdirectories.

cp *–r source destination* – copy recursively all directories under source to destination

Exercise 9

```
$ pwd
# you should be in new_directory
$ cp inner_directory also_inner
```

You can not immediately copy a directory. See the previous slide.

Poking around

How much space do I have?

quota – command to see all quotas for your directories are, if any.

How much space am I taking up?

`du` - command to find out how much space a folder or directory uses.

`df` – display space information for the entire system

Helpful Hints on Space

Almost all commands that deal with file space will display information in Kilobytes, or Bytes. Nobody finds this useful.

Many commands will support a '-h' option for "Human Readable" formatting.

`ls -lh` - displays the working directory files with a long listing format, using "human readable" notation for space

Permissions

- Linux systems are multi-user environments where many users run programs and share data. Files and directories have three levels of permissions: World, Group, and User.

- The types of permissions a file can contain are:

Read Permissions	Write Permissions	Execute Permissions
r	w	x

(execute means “allowed to enter” for directories)

- The three levels are:

User	Group	Other (i.e., world)
u	g	o

Exercise 10

```
$ cat > textfile
Just some text
^D
$ cat textfile
$ ls -l textfile
$ chmod u-r textfile
$ cat textfile # error!
$ chmod 644 textfile
$ cat textfile
$ ls -l textfile
```

You can change permissions on a file, even affecting yourself!

Changing Permissions

- **`chmod`** – change permissions on a file or directory

- **`chgrp`** and **`chown`** – change group ownership to another group (only the superuser can change the owner)

  - Both options support '-R' for recursion.

File Redirection

- Oftentimes we want to save output (stdout) from a program to a file. This can be done with the 'redirection' operator.

  ```
  myprogram > myfile
  ```

  using the '>' operator we redirect the output from myprogram to file myfile

- Similarly, we can append the output to a file instead of rewriting it with a double '>>'

  ```
  myprogram >> myfile
  ```

  using the '>' operator we append the output from myprogram to file myfile

Exercise 11

```
$ cat textfile
# the file already has content
$ cat > textfile
One line
And another
And more
^D
$ cat textfile
$ cat >> textfile
Let's add a line
^D
```

Output redirection can overwrite old content, or add to it

File display

- You have seen “cat”, but that is not great for long files.
- Better options:
  - less (also “more”): display page by page
  - head : initial part
  - tail : last part

Exercise 11

```
# get text from
# http://www.lipsum.com/feed/html
# put in a file "lorem"
$ less lorem
$ head lorem
$ tail lorem
```

Can you figure out how to display more or fewer lines?

Pipes

- Using a pipe operator '|' commands can be linked together. The pipe will link the standard output from one command to the standard input of another.

- Helpful for using multiple commands together

  example: `ls -1 ./* | wc -l`

Exercise 12

```
$ ls -l
$ ls -l | wc -l
$ cat textfile # use exercise11
$ grep a textfile
$ grep a textfile | wc -l
$ cat textfile | tr a-z A-Z
```

Output redirection can overwrite old content, or add to it

Exercise: can you get the output of that last command into a file?

grep

- man grep

```
grep [options] PATTERN [FILE...]
```

- grep searches the named input FILEs (or standard input if no files are named, or the file name – is given) for lines containing a match to the given PATTERN. By default, grep prints the matching lines.

Exercise 13

```
$ grep a textfile
$ grep -i a
```

Case insensitive

Regular Expressions

Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems. -- Jamie Zawinski

What are Regular Expressions?

- A concise grammar for doing pattern matching in strings or text.
- Often abbreviated as RegEx.
- A regular expression, often called a pattern, is an expression that describes a set of strings.
- They are typically used to give a concise description of a set, without having to list all elements.
- For example the three strings “Handel”, “Händel”, and “Haendel” can be described by the pattern:

```
H(ä|ae?)ndel
```

Why Should You Care About RegEx?

- If you use Linux, they are everywhere!
- **grep/egrep**: find strings in text files that match a pattern.
- **sed**: find pattern in text and filter or transform into something else.
- **find**: find files or directories
- **awk**: data extraction and reporting tool
- **bash**: shell scripting
- **ls**: list files

RegEx Basics

| Alternation

? Optional

* Zero or more

+ One or more

Exercise 14

```
$ grep e textfile
$ grep e. textfile
$ grep e* textfile
$ grep "h|n"e textfile # uh oh
$ grep "h\|n"e textfile
```

Search for “e”,
“e” plus one character,
“e” plus zero or more chars

Remember that “|” means something?
So now you have to “escape” it

Regular Expressions

- **Boolean "or"**
  - A vertical bar seperates alternatives
  - For example, `gray|grey` can match "gray" or "grey"
- **Grouping**
  - Parentheses are used to define the scope and precedence of the operator (among other uses). For example, `gray|grey` and `gr(a|e)y` are equivalent patterns which both describe the set of "gray" and "grey"

Regular Expressions

- **Quantification**
  - A quantifier after a token (such as a character) or group specifies how often that preceding element is allowed to occur. The most common quantifiers are the question mark `?`, the asterisk `*`, and the plus sign `+`

Regular Expressions - ?

- The question mark indicates there is zero or one of the preceding element.
- Example:

`colou?r` will match both “color” and “colour”

Regular Expressions - *

- The asterisk indicates there are zero or more of the preceding element.
- Example:

`ab*c` matches “ac”, “abc”, “abbc”, “abbbc”, etc...

Regular Expressions - +

- The plus sign indicates that there is one or more of the preceding element.
- Example:

`ab+c` matches “abc”, “abbc”, “abbbc”, etc... But NOT “ac” as in the previous example.

Regular Expression Meta Characters

- `.` (full stop)
  - Matches any single character

    `a.c` matches “abc”, etc.,

  - But `[a.c]` matches only “a”, “.”, or “c”.
- `[]`
  - A bracket expression matches a single character that is contained within the brackets.

    `[abc]` matches “a”, “b”, or “c”

  - Can use ranges
    - `[a-z]`
    - `[0-9]`

Regular Expression Meta Characters

- `[^ ]`
  - Matches a single character that is not contained within the brackets.

  `[^abc]` matches any character other than “a”, “b”, or “c”
- `^`
  - Matches the starting position within the string

  `^foo` matches lines that start with foo

Regular Expression Meta Characters

- `$`
  - Matches the ending position of the string or the position just before the string-ending newline.

  `at$` matches things like “hat” or “cat” but only if they are at the end of the line.
- `^`

  Matches the beginning position of a string or line

Exercise 15

```
$ grep "e$" textfile
$ grep –I "^o" textfile
```

Search for characters at the beginning or end of a line

Exercise 16

```
$ mkdir p
$ cd p
$ touch s sk ski skiing skill
$ ls
$ ls ski*
$ ls | grep "ski*"
```

Annoyingly, regexps do not always behave the same:

Shell: “i*” means i-followed-by-any

Grep: “i*” means any-number-of-i’s

awk

- man awk
  - Pattern scanning and processing language
- `print`
  - This displays the contents of the current line. In AWK, lines are broken down into fields, and these can be displayed separately:
- `print $1`
  - Displays the first field of the current line
- `print $1, $3`
  - Displays the first and third fields of the current line, separated by a predefined string called the output field separator (OFS) whose default value is a single space character.

awk

- File example.txt contains:

```
foo1 bar baz
foo2 barr bazz
foo3 barrr bazzz
foo4 barrrr bazzzz
```

- We would like to print only the 1st column

```
$ cat example.txt | awk '{print $1}'
foo1
foo2
foo3
foo4
```

sed

- man sed
  - Stream editor for filtering and transforming text

```
sed -I 's/foo/bar/g' ./myfile.txt
```

  - The above command will search for all instances of “foo” in the file “myfile.txt” and replace it with “bar”

Exercise 17

```
$ sed s/And/Or/ textfile
```

Enclose in single quotes if there are dangerous characters

Unix Commands

Unix commands

- Unix commands, even most system commands, are just files with executable permission
- Use “which” to find out where your command is
- Commands need to be on your $PATH

Exercise 18

```
$ which ls
$ file /bin/ls # or where it was
$ file `which ls`
# if you're on stampede:
$ file `which mpicc`
$ echo $PATH
```

Find out where a command is and what sort of file it is

Can you find system commands on your path?

What the system is up to

- **top** – show a detailed, refreshed, description of running processes on a system.
- **uptime** – show the system load and how long the system has been up.
- '**load**' is a number based on utility of the cpu's of the system. A load of 1 indicates full load for one cpu.
- **whoami** – what's your login name; convenient if you have multiple accounts
- **hostname** – what machine are you on?
- **finger** – find other users

```
login1$ uptime
 13:21:28 up 13 days, 20:12, 23 users,  load average: 2.11, 1.63, 0.91
```

Exercise 19

```
$ ps
$ ps guwax
```

Exercise: how many processes belong to you? How many to root?

Make your own unix commands: aliases

- An alias is a synonym for a command
  `alias ls='ls –F –color=none'`
- Type '`alias`' to see what aliases are defined

Make your own unix commands: scripts

- You can make a shell script by putting commands in file
- And…
  - Tell the shell what “command interpreter” to use
  - Set executable permission
  - And make sure it is on your path

Exercise 20

```
$ cat > work.cmd
#!/bin/bash
echo "It works!"
^D
$ work.cmd # does not work
$ ./work.cmd # different reason
$ # how do you fix this?
$ mkdir -p ~/bin
$ mv work.cmd ~/bin
$ ~/bin/work.cmd
```

Why doesn't it work?
See the previous slide.

Background and Foreground

- ^z (control+z) suspends the active job
- `bg` – resumes a suspended job in the background and returns you to the command prompt
- `fg` – resumes a background job in the foreground so you can interact with it again
- `fg %2` – resumes a specific background

Exercise 21

```
$ cat > date.cmd
#!/bin/bash
while [ 1 ] ; do
  sleep 2
  date
done
^D
$ # make executable and run
$ ^Z
$ # fg, bg, ps
$ ^C
```

This will run forever.

Use ^Z to stop it,
use bg and fg to juggle

Use ps to see what's running

Advanced Program Options

- Often we must run a command in the background with the ampersand '&' character

  ```
  command -options &
  ```

  runs command in background, prompt returns immediately

- Match zero or more characters wildcard '*'

  ```
  cp * destination
  ```

  copy everything to destination

  This option can get you into trouble if misused

Editing and Reading Files

- **emacs** vs. **vim**
  - Among the largest 'nerd battle' in history known as the "Editor War"[0].
  - **emacs** relies heavily on key-chords (multiple key strokes), while **vim** is mode based. (editor mode vs. command mode)
  - **vim** users tend to enter and exit the editor repeatedly, and use the Linux shell for complex tasks, whereas **emacs** users usually remain within the editor and use **emacs** itself for complex tasks
- **less**
  - If you only need to read a file (not edit it), programs like less give you "read only" access and a simplified interface

[0] - http://en.wikipedia.org/wiki/Editor_war

Searching for files

A large majority of activity on Linux systems involve searching for files and information.

find – utility to find files

```
login1$ find . -name foobar
./test_dir/foobar
login1$ cat ./test_dir/foobar
=======
*
    This is the file I searched for!
*
=======
```

Input and Output

- Programs and commands can contain an input and output. These are called 'streams'. Linux programming is oftentimes stream based.
  - Programs also have an error output. We will see later how to catch the error output.

STDIN – 'standard input,' or input from the keyboard

STDOUT – 'standard output,' or output to the screen

STDERR – 'standard error,' error output which is sent to the screen.

Other Useful Commands

`head file.txt`

– prints the first 10 lines of a file

`tail -n 5 file.txt`

– prints the last 5 lines of a file

`history`

– prints your command history

example:

Try These Commands

```
history | grep "ls"
```

- This command will pipe the output of “history” to the command “grep” which will search for instances of ls

Bash “Reverse Intelligent Search”

- This bash keybinding is so important for developer’s productivity it’s well worth repeating over and over again.
- Search your history in bash, just press ***Ctrl-R*** to do a reverse intelligent search
- Once you have found the command you have a few options:
  - Run it verbatim – just press ***ENTER***
  - Edit it before running – use the arrow keys to navigate to the point you want to edit
  - Cycle through other commands that match the letters you’ve typed – press ***Ctrl-R*** successively
  - Quit the search and back to the command line empty-handed – press ***Ctrl-G***

UNIX vs. Windows files

- File formats are different between the two operating systems
- Use the UNIX command dos2unix to convert files – especially script files - created on Windows, so they will work on UNIX

File Transfers

- Both **scp** and **rsync** are simple file transfer tools.
- scp usage:
  - scp [options] SOURCE DESTINATION
  - Example:

```
login1$ scp myfile.txt jlockman@stampede.tacc.utexas.edu:
```

  - This will copy the file "myfile.txt" to Stampede in my home folder (/home1/00944/jlockman)
  - You could also provide the full path

```
login1$ scp myfile.txt jlockman@stampede.tacc.utexas.edu:/work/00944/jlockman/foo
```

File Transfers

- rsync usage:
  - rsync [options] SOURCE DESTINATION
  - Example:

```
login1$ rsync myfile.txt jlockman@stampede.tacc.utexas.edu:
```

  - This will copy the file “myfile.txt” to Stampede in my home folder (/home1/00944/jlockman)

  - You might also rsync an entire directory

```
login1$ rsync -av ./foo/ jlockman@stampede.tacc.utexas.edu:~/foo
```

Conclusions

- There is a lot of information presented here, don't become overwhelmed.
- Linux is a full featured OS with several useful tools right out of the box.
- Pick up a book on regular expressions to generate better queries of text files

How to Get Help
Before we go further…

- Read the Manual.
  - **man command**
  - **man [section] command**
  - **man –k keyword** **(search all manuals based on keyword)**
- Most commands have a built-in manual, even the **man** command!
- Commands without manuals have help too, with **–h**, **--help**, or **/?** option.

Shell “Preferences”

- Shells execute startup scripts when you login
- You can customize these scripts with new *environment variables* and *aliases*
  - For bash: ~/.profile
  - For tcsh: ~/.cshrc

Victor Eijkhout

eijkhout@tacc.utexas.edu

For more information:

www.tacc.utexas.edu