Introduction to Linux, Commands, Hands On

Spring 2017

Victor Eijkhout and Charlie Dey

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 UNIX manual, even the **man** command!

- Commands without manuals have help too, with **–h**, **--help**, or **/?** option.

How to Get Help
The Manual

- The manual pages are divided into eight sections depending on type of command.
  - 1 commands and applications
  - 2 system calls
  - 3 C library functions
  - 4 special files
  - 5 file formats
  - 6 games
  - 7 misc.
  - 8 system administration utilities

Conventions for this lecture

- Commands will be in bold, options will be in italics.

  `command –`*`arguments`*

- In helpfiles and manuals, commands will have required input and option input
- **cp** *[OPTION] source destination*
  - Optional arguments are in brackets, required arguments are not.
- **cp** –R or **cp** - -recursive
  - Short options '-', long options '- -'

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

Linux Accounts

- A username is (typically) a sequence of alphanumeric characters of length no more than 8:
  - *eg. jlockman* or *istc00, istc01, …*
- The username is the primary identifying attribute of your account
- the name of your home directory is usually related to your username:
- *eg. /home/jlockman*

Linux Accounts

- A password is a secret string that only the user knows (not even the system knows it)
- When you enter your password the system encrypts it and compares to a stored string
- passwords are (usually) no more than 8 characters long.
- It's a good idea to include numbers and/or special characters (don't use an english word, as this is easy to crack)

Linux Accounts

- Linux includes the notion of a "group" of users
- A Linux group can share files and active processes
- Each account is assigned a "primary" group
- The *groupid* is a number that corresponds to this primary group
- In Linux-speak, groupid's are knows as *GID's*
- A single account can belong to many groups (but has only one primary group)

Interacting with the Shell

Type a command (**`ls`**) at the prompt (**`login3$`**) and press ENTER

Example: **`login3$ ls`**

- Shell starts a new process for executing the requested command , the new process executes the command and the shell displays any output generated by the command
- When the process completes, the shell displays the prompt and is ready to take the next command
- Specific information is passed to the command via more arguments
- The shell is killed by “exit” or CTRL-D

```
login3$ exit
logout
```

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

A Bit about Directories and Files

- Linux/Unix stores **files** in **directories**.
- Directories are hierarchical—an inverse-tree organization. So, a given directory can have many directories in it (as well as many files)
  - This is like a folder having folders in it, in OS X and Windows (in fact, it's exactly the same thing)
- The 'root' (top) directory in Linux/Unix is "***/***"
  - every other directory is /something(/something…)
  - forward slash is used to separate directory and file names
  - User 'home' directories are often something like:
    - /home/username, or
    - /users/username

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: Unix is case-sensitive
  - So, “texas-fight” is different than “Texas-Fight”

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*

Directories

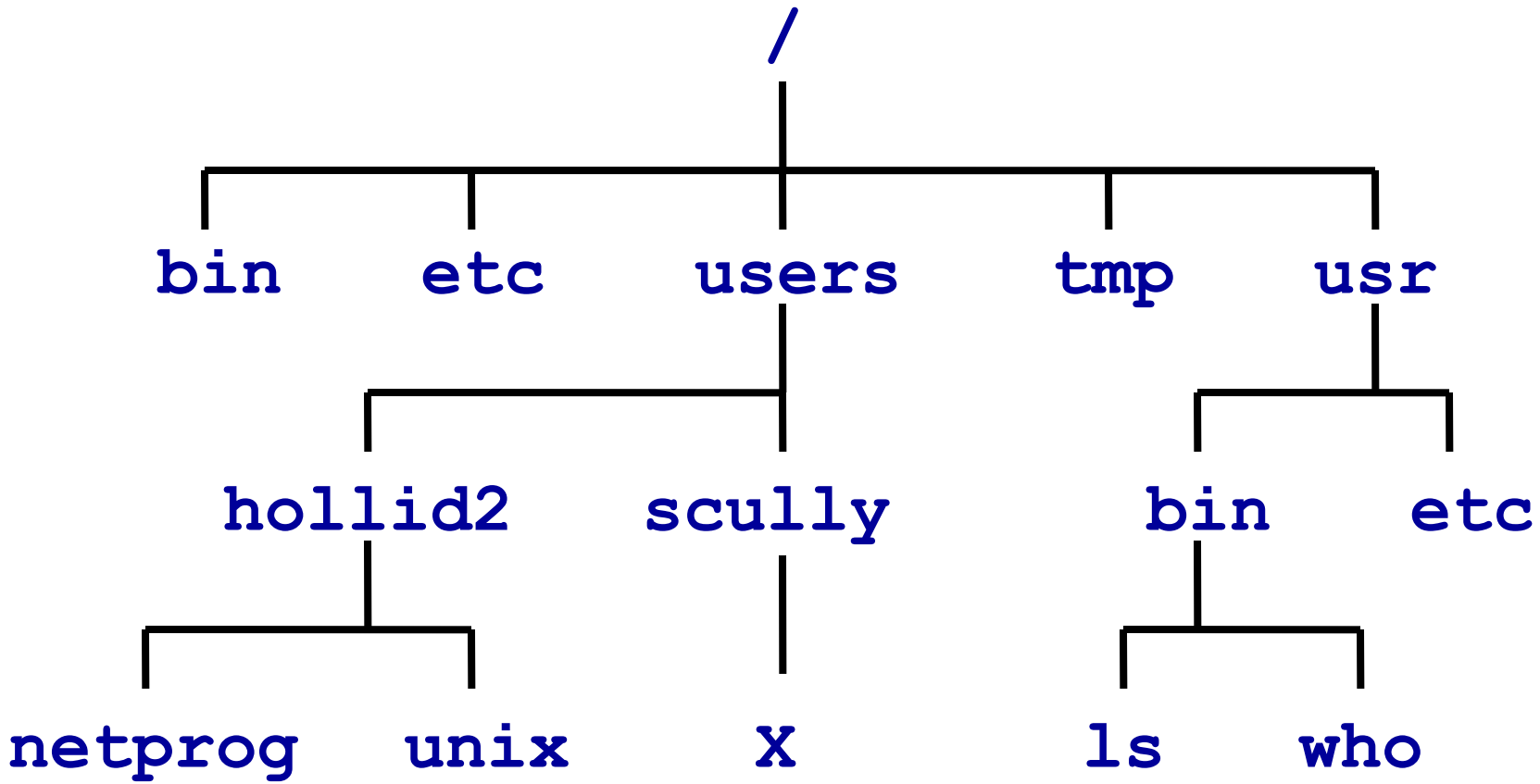
What is a *working directory*?

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

pwd – view the path of your working directory

ls – view your working directory

Unix Filesystem (an upside-down tree)



Finding your home

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

cd

cd *~jlockman*

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.

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

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!

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

Poking around in $HOME

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

- The *NIX 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

Permissions Cont.

- File permissions are arranged in three groups of three characters.
- In this example the owner can read & write a file, while others have read access

User (owner)	Group	Others (everyone else)
rw-	r--	r--

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.

All About Me

Every userid corresponds to a unique user or system process

whoami – returns the userid of the current user

passwd – change password

What is my group? – G-81769

```
slogin1$ ls -l ~jlockman
total 12
-rwxr--r--  1 jlockman G-81769 18 Jan 27 12:04 bar
-rwxr--r--  1 jlockman G-81769 13 Jan 27 12:04 baz
-rwxrwxrwx  1 jlockman G-81769 37 Jan 27 12:04 foo
-rwxr-xr-x  1 jlockman G-81769  0 Jan 27 12:05 someFile
```

Everyone else

- **who** – show all other users logged in
- **finger** – show detailed information about a user

```
slogin1$ finger jlockman
Login: jlockman                         Name: John Lockman III
Directory: /home/00944/jlockman             Shell: /bin/bash
On since Tue Jan 27 13:15 (CST) on pts/0 from lockman-d620.tacc.utexas.edu
New mail received Sat Aug 30 15:34 2008 (CDT)
     Unread since Tue Aug 26 12:34 2008 (CDT)
Plan:
High Performance Computing Specialist
Texas Advanced Computing Center
jlockman@tacc.utexas.edu
```

Basic Commands (1)

- To print the name of the current/working directory, use the **pwd** command

```
login4$ pwd
/share/home/01698/rauta
```

- To make a new directory, use the **mkdir** command

```
login4$ mkdir ssc322
```

- To change your working directory, use the **cd** command

```
login4$ cd ssc322
```

Basic Commands (2)

- To create a new file use the **vi** command

```
login4$ vi test.txt
```

  - Press **i** to start inserting text
  - Type some text: **Hello Class 322**
  - To save and quit, press Esc key, and enter **:wq!**

    (press the enter key after typing **:wq!**)
  - To quit without saving, press Esc key if in insert mode, and enter **:q!**

- To display the contents of the file, use the **cat** short for concatenation) command

```
login4$ cat test.txt
```

Basic Commands (3)

- To list the contents of a directory, use the **`ls`** command

```
login4$ ls
```

- To see all files and directories, including hidden ones use the **`-a`** flag with the **`ls`** command. Hidden files have a "**.**" infront of them

```
login4$ ls -a
```

Note: your current working directory can be checked by using the `pwd` command.

Basic Commands (4)

- To copy contents of one file to another, use the **cp** command

```
login4$ cp test.txt copytest.txt
login4$ cp test.txt test3.txt
```

One more example:

```
login4$ mkdir junk
login4$ cp test.txt ./junk/test2.txt
```

(The command above copies a file to the sub-directory **junk**)

```
login4$ cd junk
login4$ ls
login4$ cd ..
```

- To go a level up from the current working directory

```
login4$ cd ..
```

Basic Commands (5)

- To remove a file, use the **rm** command

```
login4$ rm test2.txt
```

- To remove a directory, use the **–r** option with the **rm** command

```
login4$ rm –r junk2
```

- You can also use the **rmdir** command to remove an empty directory

```
login4$ rmdir junk2
```

Note: rmdir command does not have –r option

Basic Commands (6)

- A file can be renamed by moving it. The same can be achieved by using the **mv** command

```
login4$ mv test3.txt newtest3.txt
```

- Use the **man** command to get more information about a command – it is like using help in Windows

```
login4$ man rmdir
```

- Use the **diff** command to see the differences in two files

```
login4$ diff test.txt newtest3.txt
```

Basic Commands (7)

- Previously executed commands in a shell can be viewed by using the **`history`** command. For example:

```
login4$ history
  1  man ls
  2  ls -ltr
  3  ls -l -t -r
  4  ls -ltr
  5  history
```

Basic Commands (8)

- If the contents to display are more than one page, you could use the **`more/less`** command for paging through text a screenful at a time

```
login4$ more test.txt
login4$ less test.txt
```

(**`less`** allows both fwd and bwd movement)

Basic Commands (9)
Creating a tarball

- TAR (Tape Archive) command bundles files and sub-directories together and creates an archive (known as tar file or tarball)

- To create a tarball of all the files and sub-directories in the directory ssc329 that you created in Exercise 1, use **c** flag:

```
tar -cvf mytar.tar *
```

- To extract the contents of a tar file use **x** flag:

```
login1$ tar -xvf mytar.tar
```

What everyone else 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.

```
slogin1$ uptime
 13:21:28 up 13 days, 20:12, 23 users,  load average: 2.11, 1.63, 0.91
```

Working With Programs

- Commands or programs on the system are identified by their filename and by a process ID which is a unique identifier.
  - ps – display process information on the system
  - kill pid – terminates the process id
  - ^c (control+c) terminates the running program
  - ^d (control+d) terminates your session.
- Only you and the superuser (root) has permissions to kill processes you own.

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

Input and Output

- Programs and commands can contain an input and output. These are called ‘streams’. Unix 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.

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

Input Redirection

- Input can also be given to a command from a file instead of typing it to the screen, which would be impractical.

```
cat programinput > mycommand
```

- This command series starts with the command '`cat`' which prints a file to the screen. programinput is printed to stdout, which is redirected to a command mycommand

Redirecting stderr

- Performing a normal redirection will not redirect sdterr. In Bash, this can be accomplished with '2>'

```
command 2> file1
```

- Or, one can merge stderr to stdout (most popular) with '2>&1'

```
command > file 2>&1
```

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.

- Very helpful for searching files

Searching

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

find – utility to find files

grep – the best utility ever written for Unix, searches for patterns inside files and will return the line, if found

```
slogin1$ find . -name foobar
./test_dir/foobar
slogin1$ cat ./test_dir/foobar
=======
*
    This is the file I searched for!
*
=======
```

Compression using gzip

- `slogin1$ du -h bigfile`
- `32Kbigfile`
- `slogin1$ gzip bigfile`
- `slogin1$ du -h bigfile.gz`
- `4.0K  bigfile.gz`

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

Using tar to create compressed files

- Tar will create compressed files for you
  - **tar** *–czvf mytarfile.tar.gz directory*
    - creates a compressed file named *mytarfile.tar.gz* containing all of the files in the directory *directory*
  - **tar** *–xzvf mytarfile.tar.gz*
    - uncompresses all directories and files inside the file *mytarfile.tar.gz* into the working directory

Connecting to Another Machine

- Secure Shell vs Restricted Shell
  - **ssh** is an encrypted remote login program that is 'secure' to trust across non secure networks.

- **ssh** *userid@hostname*

Copying Files to Remote Hosts

- copy local file *lfile* to *rfile* on remote machine *rsys*
  - **scp** *lfile rsys*:*rfile*
    - **-p** preserves modification time, access time and mode from original
      - **scp -p** *lfile rsys*:*rfile*

- copy *rfile* from remote machine *rsys* to local file *lfile*
  - **scp -p** *rsys*:*rfile lfile*

Running Commands on a Remote Host

- Commands can be executed on a remote host with **ssh**
- **ssh** *userid@hostname* "ls"
  - Run ls on remote host hostname

My Environment

- View all system variables by the command '**env**'
- Depending on shell, startup commands can be managed with the files .profile for bash and .cshrc with c shell

Basic Shell Scripts

- Many times it is helpful to create a 'script' of commands to run instead of typing them in individually. Scripts can be made to aid in post-processing, system administration, and automate menial tasks

- #!/bin/bash
  - First statement inside a script, will list which shell to run this script in

- # - says what will follow is a comment and not to execute

Basic Shell Scripts
Variables

- By convention system variables are capitalized
  - HOME – location of the home directory
  - OLDPWD – location of the previous working directory
  - PATH – locations to look inside for executable files
- Setting system variables differs by shell. bash uses **export**, csh uses **setenv**
- User defined variables in scripts are lower-case by convention
  - myvariable=10
    - sets myvariable to 10
  - **echo** $myvariable
    - prints myvariable

Basic Shell Scripts
Conditionals

```
if condition
  then
      condition is zero (true - 0)
   execute all commands up to else
  statement
  else
      if condition is not true then
  execute all commands up to fi
fi
```

Basic Shell Scripts
Performing Loops

Loops are statements that are repeated until the conditions are met.

```
for { variable name } in { list }
do
   execute one for each item in the list until the
   list is not finished (And repeat all statement
   between do and done)
done

for i in 1 2 3 4 5
do
   echo "Welcome $i times“
done
```

Basic Shell Scripts
Putting it Together

```
#!/bin/bash
#my first script
#scp replacement

remotefile=mydata
localfile=mydata
myserver=dstanzi@lonestar.tacc.utexas.edu
mylsinfo=`ssh $myserver ls $remotefile 2>&1`
ismissing=`echo $mylsinfo | grep ERROR`
if [ "$ismissing" ]
then
   echo "$remotefile not found! Exiting!"
else
    ssh $myserver  -n "cat < $remotefile" > $localfile
fi
```

Basic Shell Scripts
More…

- `mylsinfo=`ssh $myserver ls $remotefile 2>&1``
  - `Backticks` are used to place output from a command into a variable
- `if [ "$ismissing" ]`
  - Is $ismissing set (has a value)? If so then the expression is true, otherwise false

Text Editing

- To be productive in this class, you'll need to be able to use a text editor, to write and edit your code.
  - Microsoft Word is not a Text Editor ☺
- You have a couple of options:
  - Edit locally on your machine, and transfer files (via scp/ sftp) to the machine you run your program on.
    - Ultimately, you will find this is annoying
  - Learn to use a text editor on the UNIX system

Text Editors on (most) *nix Systems

- Pico/Nano
  - Very simple to use, you can learn it in 10 minutes.
  - Usable, will get you through class.
  - Not very sophisticated.
- Vi or EMACS
  - Steeper learning curve (the first 10 minutes will be painful).
  - Much more powerful (the next 30 years will be smoother).
  - Choosing between Vi or EMACS is like picking a religion.
- Let me give you a short demo of Nano and Vi…

References

- http://code.google.com/edu/tools101/linux/basics.html#the_command_line
- http://www.tacc.utexas.edu/documents/13601/118360/LinuxIntro_HPC_09+11+2011_hliu.pdf
- http://www.cis.uab.edu/courses/cs333/spring2005/
- http://www.med.nyu.edu/rcr/rcr/nyu_vms/Unix-Editors.html
- http://www.shelldorado.com/articles/mailattachments.html