Introduction to Linux

Daniel Trahan
daniel.trahan@colorado.edu

www.colorado.edu/rc

Slides available for download from:
https://github.com/ResearchComputing/HPC_short_course_summer_2018

Outline

- What is Linux?
- Why use Linux?
- What happens when you log in?
- Shells and environment
- Commands
- Files / Directories / Filesystems
- Processes
- More about shells

What is Linux?

- Part of the Unix-like family of operating systems.
- Started in early ‘90s by Linus Torvalds.
- Typically refers only to the kernel with software from the GNU project and elsewhere layered on top to form a complete OS. Most is open source.
- Several distributions are available – from enterprise-grade, like RHEL or SUSE, to more consumer-focused, like Ubuntu.
- Runs on everything from embedded systems to supercomputers.

Why Use Linux?

- Default operating system on virtually all HPC systems
- Extremely flexible
- Not overbearing
- Fast and powerful
- Many potent tools for software development
- You can get started with a few basic commands and build from there

How do you login?

- To a remote system, use Secure Shell (SSH)
- From Windows
  - Non-GUI SSH application: Windows PowerShell
  - GUI SSH application: PuTTY
  - Putty is prefered method.
- From Linux, Mac OS X terminal, or Windows GUI such as Cyberduck, PuTTY or Gitbash – ssh on the command line

  `ssh username@tutorial-login.rc.colorado.edu`
- Once you are logged on, type the following:

  `ssh scompile`

Useful SSH options

- `-X`
  - Allows X-windows to be forwarded back to your local display
- `-o TCPKeepAlive=yes`
  - Sends occasional communication to the SSH server even when you're not typing, so firewalls along the network path won't drop your "idle" connection

What happens when you login?

- Login is authenticated (password or key)
- Assigned to a tty
- Shell starts
- Environment is set up
- “Message of the Day” prints
- Prompt

What identifies a Linux user?

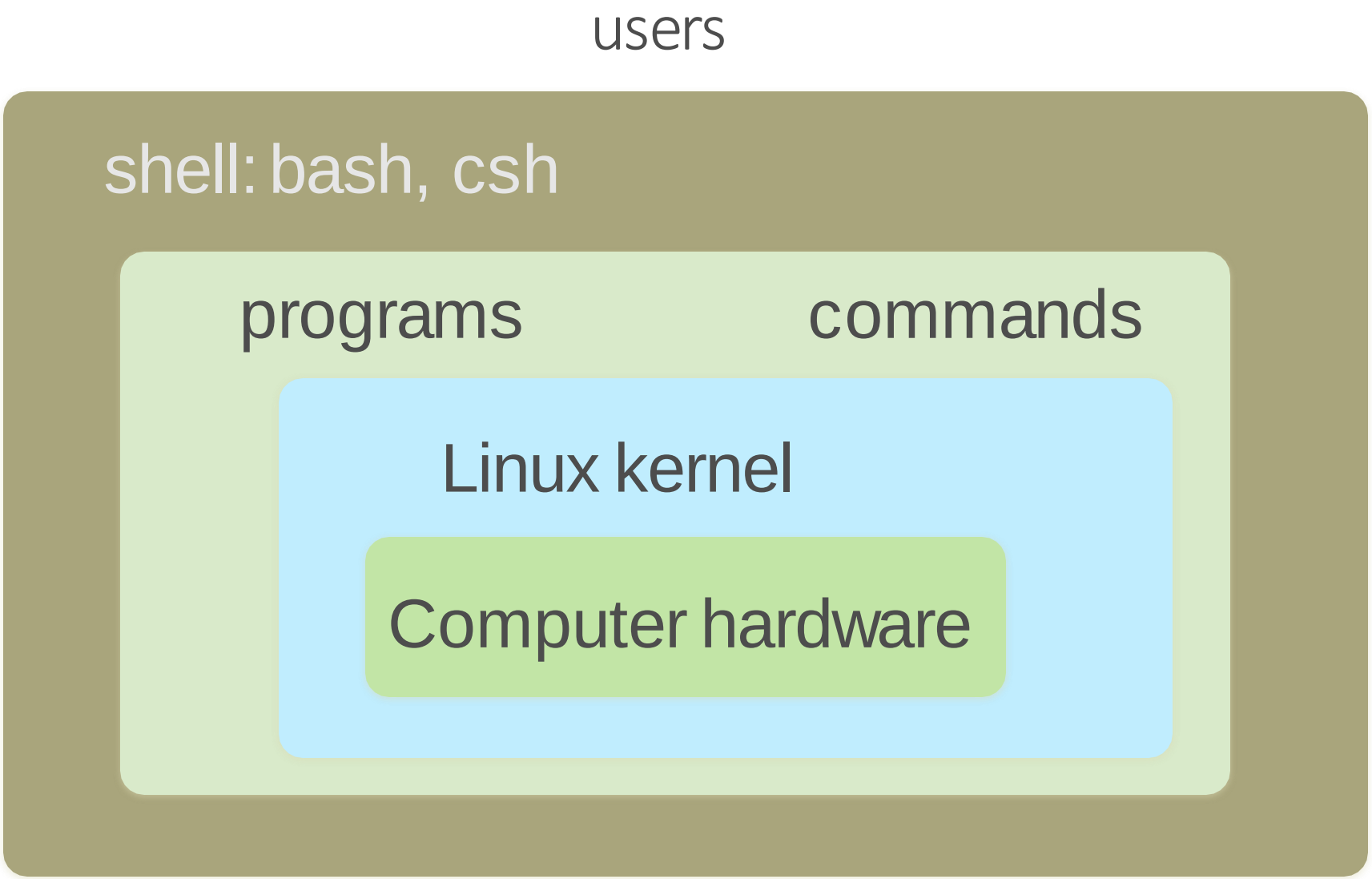
- Username / UUID
- Group / GID
- Password (or other authentication info)
- GECOS
- Default shell
- Home directory (ie, home "folder" on disk)

Shells

The shell parses and interprets typed input, passes results to the rest of the OS, returns response as appropriate

- Bourne (sh) – early and rudimentary
- Bourne-again (bash) – has many user-friendly extensions; default in Linux
- C (csh) – has C-like syntax
- T (tcsh) – extended version of csh
- Korn (ksh) – early extension of Bourne; was heavily used for programming
- Z (zsh) – includes features of bash and tcsh

Shells


users
shell: bash, csh
programs
commands
Linux kernel
Computer hardware

Shell features

- Tab completion
- History and command-line editing
- Scripting and programming
- Built-in utilities

Anatomy of a Linux command

- command [flags] [flag arguments] [target(s)]
- `tar -c -f archive.tar mydir`
- Flags may not mean the same thing when used with different commands
- The same command may have different flags in different kinds of Unix (esp. Linux vs BSD)
- Case is important!
- Order of flags may be important

The most important Linux command:

man

man <command>

man -k <keyword>

File and directory related commands

pwd – prints full path to current directory
cd – changes directory; can use full or relative path as target
mkdir – creates a subdirectory in the current directory
rmdir – removes an empty directory
rm – removes a file (`rm -r` removes a directory and all of its contents)
cp – copies a file
mv – moves (or renames) a file or directory
ls – lists the contents of a directory (`ls -l` gives detailed listing)
chmod/chown – change permissions or ownership
df – displays filesystems and their sizes
du – shows disk usage (`du -sk` shows size of a directory and all of its contents in KB)

Process and Program related commands

ps – lists processes (`ps -ef` lists all running processes)

top – shows processes currently using the CPU

kill – sends a signal to a process (kills process by default). Target is Process-ID; found in 2nd column of `ps -ef` output.

jobs – shows jobs currently in background

time – shows how much wall time and CPU time a process has used

nice – changes the priority of a process to get CPU time

File-viewing commands

less – displays a file one screen at a time

cat – prints entire file to the screen

head – prints the first few lines of a file

tail – prints the last few lines of a file (with -f shows in realtime the end of a file that may be changing)

diff – shows differences between two files

grep – prints lines containing a string or other regular expression

tee – prints the output of a command and also copies the output to a file

sort – sorts lines in a file

find – searches for files that meet specified criteria

wc – count words, lines, or characters in a file

Environment

- Set up using shell and environment variables
  - shell: only effective in the current shell itself
  - environment: carry forward to subsequent commands or shells
- Set default values at login time using `.bash_profile` (or `.profile`). Non-login interactive shells will read `.bashrc` instead.
- `var_name[=value]` (shell)
- `export VAR_NAME[=value]` (environment)
- `env` (shows current variables)
- `$VAR_NAME` (refers to value of variable)

Useful variables

- `PATH`: directories to search for commands
- `HOME`: home directory
- `DISPLAY`: screen where graphical output will appear
- `MANPATH`: directories to search for manual pages
- `LANG`: current language encoding
- `PWD`: current working directory
- `USER`: username
- `LD_LIBRARY_PATH`: directories to search for shared objects (dynamically-loaded libs)
- `LM_LICENSE_FILE`: files to search for FlexLM software licenses

Exercise 1

1. Print your current `PATH` and `LD_LIBRARY_PATH` environment variables
2. Type

   `which icc`

   to try to find the path to the Intel C Compiler command
3. Type

   `module load intel`

   to set up your environment to use the Intel compilers
4. Print your current `PATH` and `LD_LIBRARY_PATH` environment variables again. What has changed? What does `which icc` say now? Why?

The Linux Filesystem

- System of arranging files on disk
- Consists of directories (folders) that can contain files or other directories
- Levels in full paths separated by *forward* slashes, e.g.
  /home/nunez/scripts/analyze_data.sh
- Case-sensitive; spaces in names discouraged
- Some shorthand:

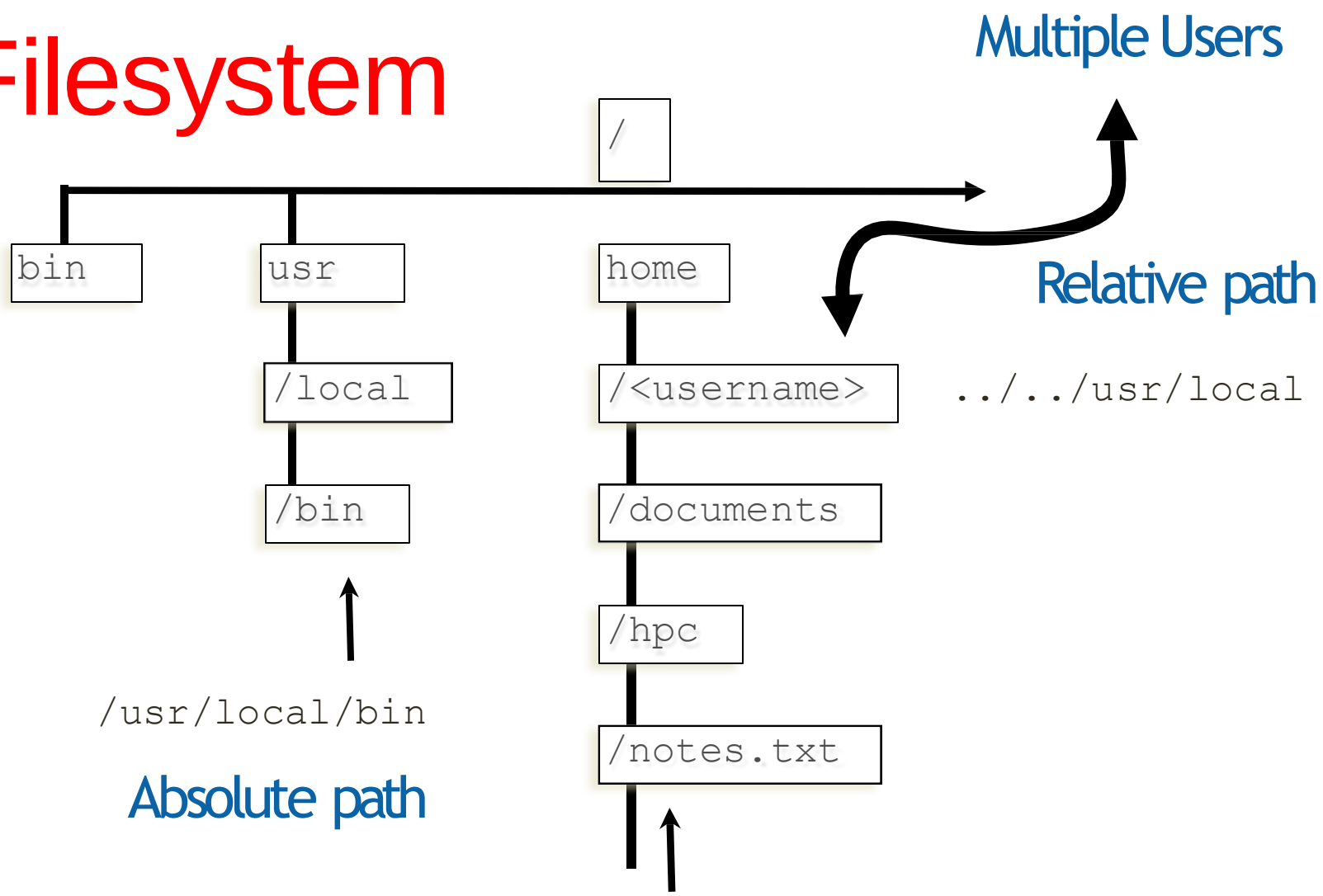
. (the current directory)

.. (the directory one level above)

~ (home directory)

- (previous directory, when used with `cd`)

Filesystem


Multiple Users
/
bin
usr
home
Relative path
/local
/<username>
../../usr/local
/bin
/documents
/hpc
/usr/local/bin
/notes.txt
Absolute path
/home/<username>/documents/hpc/notes.txt

Navigating the filesystem

- Examples:
  - ls
  - mkdir
  - cd
  - rm
- Permissions (modes)

Exercise 2

1. Change to your home directory
2. Change to HPC_Short_Course_Fall_2018/introToLinux
3. Print the path to your current directory
4. Print a "long" listing of the contents of this directory
5. List the contents of the "testfiles" directory without changing into that directory
6. Change into the "testfiles" directory
7. Change into the "scripts" directory using a single command
8. Change to your home directory and create a new directory (you can pick the name). How can you be sure the new directory is there? Rename the new dir.
9. Bonus: Determine how many KB are in "testfiles"

File editing

- **nano** – simple and intuitive to get started with; not very feature-ful; keyboard driven
- **vi/vim** – universal; keyboard driven; powerful but some learning curve required
- **emacs** – keyboard or GUI versions; helpful extensions for programmers; well-documented
- **LibreOffice** – for WYSIWYG

http://xkcd.com/378/

Modes (aka permissions)

- 3 classes of users:
  - User (u) *aka "owner"*
  - Group (g)
  - Other (o)
- 3 types of permissions:
  - Read (r)
  - Write (w)
  - Execute (x)

```
rwxr-xr--
```

Modes (continued)

- `chmod` changes modes:

To add write and execute permission for your group:

```
chmod g+wx filename
```

To remove execute permission for others:

```
chmod o-x filename
```

To set only read and execute for your group and others:

```
chmod go=rx filename
```

Exercise 3

1. Change directory to `~/HPC_Short_Course_Fall_2018/introToLinux/scripts`
2. Use `cat` to show the contents of `hello.sh`
3. Try to run `hello.sh` by typing its name at the command line
4. Add execute permission to `hello.sh` using `chmod`
5. Can you run it now?
6. Is there a path issue? What are two ways you could get the script to run?

Processes

- A process is a unique task; it may have threads
- Examples:
  - Foreground vs background (&)
  - jobs command
  - Ctrl-C vs Ctrl-Z ; bg
  - kill

More about shells

- Input and output redirection
  - Send output from a command to a new file with >
  - Append output to an existing file with >>
  - Use a file as input to a command with <
- Pipes: | sends output of one command to another command

```
ps -ef | grep ruprech
```

Shell Wildcards and Special Characters

- * - matches zero or more characters
- ? - matches a single character
- # - comment; rest of the line is ignored
- \ - escape; don't interpret the next character

Thank you!

Please fill out the survey!!!

http://tinyurl.com/curc-survey18

A good introductory online tutorial:

http://www.ee.surrey.ac.uk/Teaching/Unix/index.html