

# INTRODUCTION TO LINUX AND THE COMMAND LINE INTERFACE

# LINUX

- What is Linux?
  - Started out as a hobby project built from scratch using C by Linus Torvalds
  - Developed by the community to be similar to UNIX
- What about UNIX?
  - Originally an operating system
  - Modified(forked) into different variants by people
  - Currently both a trademark and a specification of the Open Group consortium
  - An OS that is certified by the Open Group may be called 'UNIX'
    - e.g. Apple OS-X, Oracle Solaris, IBM AIX, HP-UX
  - Linux is merely 'UNIX-like' → It's not certified

# LINUX

- Linux as an **open Source** software
  - Means that people are allowed to obtain the code and modify it to suit their needs
  - The code is not owned by any company
  - It is developed by the community that uses it
- Components of Linux as an operating system
  - Kernel
    - Central controller that manages hardware resources for the use of applications
  - Applications
    - Software that make requests to the kernel for resources to perform tasks
    - *Process* is a task that is loaded and tracked by the kernel
    - Applications may run 1 or more processes

# LINUX DISTRIBUTIONS

- Are operating systems that share the same Linux kernel, but differ by the set of tools , applications, and package managers that they come with
- May come readily compiled for installation on specific architectures or may be in code form
- Some common distros
  - **Red Hat** – built to be more suited for use on servers, offers paid support
  - **Open SUSE** – targetted for software developers and system administrators
  - **Debian** – community backed distribution which directly supports several non-Intel / AMD platforms
  - **Ubuntu** – most popular desktop / server distro derived from Debian
  - **Android** – uses the Linux kernel, but lacks support for many tools

# USING LINUX

- Graphical Mode



- Non- Graphical Mode

```

Ubuntu 12.04.1 LTS SYSServer tty1

SYSServer login: administrator
Password:
Last login: Fri Sep 16 13:27:44 PHT 2016 on tty1
Welcome to Ubuntu 12.04.1 LTS (GNU/Linux 3.2.0-29-generic-pae i686)

* Documentation:  https://help.ubuntu.com/

System information as of Fri Sep 16 14:50:11 PHT 2016

System load:  0.0                Processes:            64
Usage of /:   5.5% of 19.18GB    Users logged in:     0
Memory usage: 6%                IP address for eth0: 172.16.10.81
Swap usage:   0%

Graph this data and manage this system at https://landscape.canonical.com/

New release '14.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

administrator@SYSServer:~$ _

```

# THE LINUX COMMAND LINE

- Why is learning the command line important?
  - More precise control
  - Greater speed
  - Ability to automate tasks
- What is a 'shell'?
  - Interpreter that translates commands entered by the user into actions performed by the OS
  - There are many types of shells; one of the most common is the BASH shell

# THE LINUX COMMAND LINE

- What is a 'prompt'?
  - Terminal window displays a prompt when no commands are being run
  - Tells a user to enter a command


**administrator@SYSServer:~\$**



Logged in user



System name



Current directory  
(~ means home dir)

# THE LINUX COMMAND LINE

- What is a 'command'?
  - It's a software program that when executed, causes the OS to run a process
- Typical format: **command [options] [arguments]**
  - Options modify core command behavior
    - Single letter preceded by a dash e.g. "ls -l -h" or "ls -lh"
    - Full-word preceded by double dash e.g. "ls --human-readable"
  - Arguments provide additional information

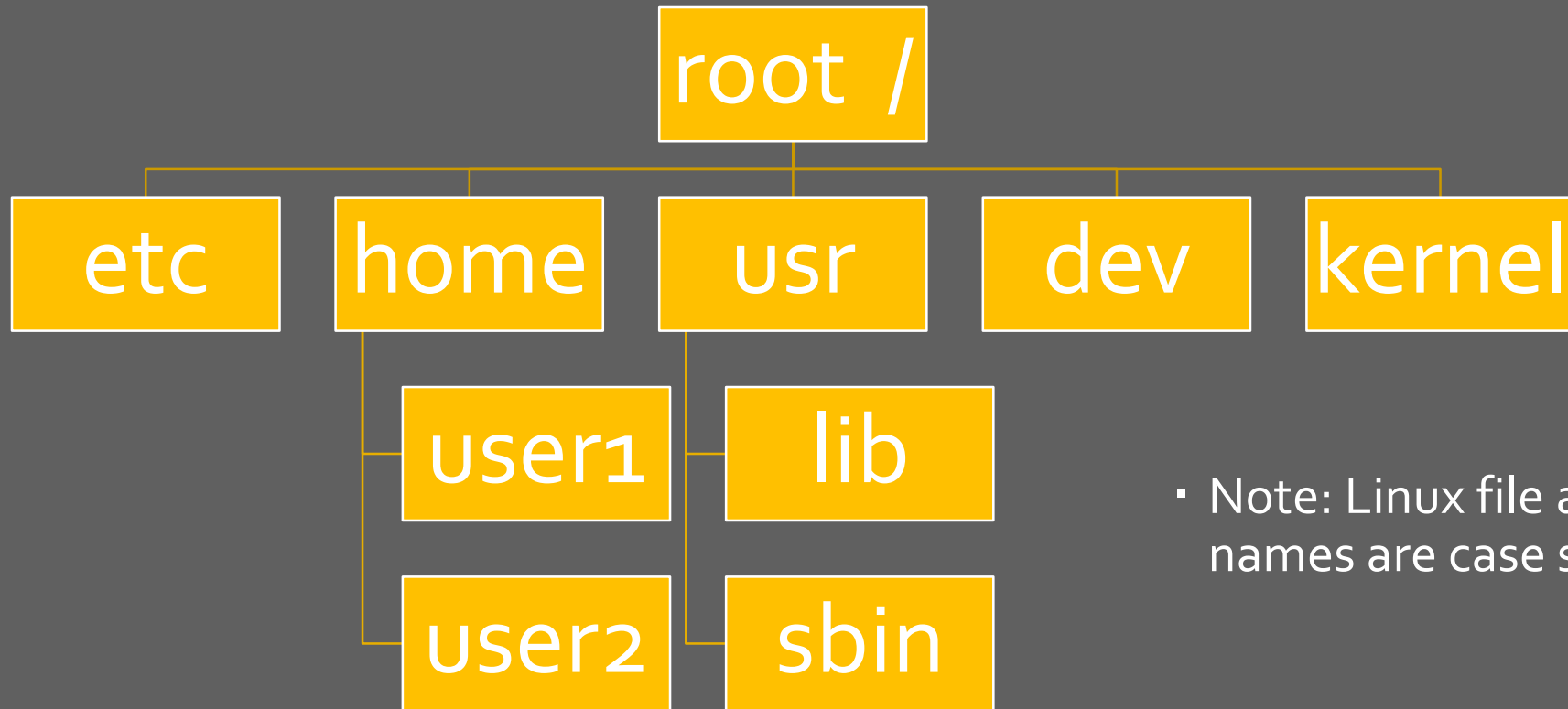


# KEY SEQUENCES

Key Sequence	Description
Ctrl-c	Terminates or aborts a process
Ctrl-d	Sends an end-of-input on the keyboard
Ctrl-s	Pauses screen output
Ctrl-q	Continues screen output
Tab	Auto-complete files and folder names
Up / Down arrows	Cycle through command history

# LINUX DIRECTORY STRUCTURE

- Linux distributions commonly follow this hierarchy:



- Note: Linux file and folder names are case sensitive

# LINUX DIRECTORY STRUCTURE

- Home Directory
  - Is the directory allocated to each user on the system normally placed under the 'Home' OS directory – e.g. /home/alice
  - A user has full control over his/her own home directory
  - OS defaults to the user home directory when logging in to the terminal or opening a shell
  - shortcut representation of home directory ` ~ `

# WORKING WITH DIRECTORIES AND FILES

## `pwd`

- Prints the directory including the path that the user is currently working in

## `cd [path]`

- Changes to a different directory specified in the path
- Path can be:
  - Absolute – pathname starts from root directory
  - Relative – pathname starts from current working directory

# WORKING WITH DIRECTORIES AND FILES

## ls

- Lists files and folders in a directory

Option	What it does
-a	Lists all contents including hidden files and folders
-l	Includes file metadata
-h	Human readable file size, normally used together with -l
-d	List current directory, normally used with -l
-R	Recursive listing
-S	List sorted by size (largest first)
-t	List sorted by modification timestamp (most recent first)

# WORKING WITH DIRECTORIES AND FILES -

- Metacharacters
  - Can be used to specify patterns to match filenames in a directory
  - Also referred to as “wildcards”
  - Are interpreted by the shells first before running a command

Character	Use	Example
*	represents zero or more of any character	Match files that start with a 't' - <code>ls /etc/t*</code>
?	represents a single character	Match any 3-letter file that starts with a 't' - <code>ls /etc/t??</code>
[ ]	Represents range of characters	Match files that start with either a 't' or 'y' - <code>ls /etc/[ty]*</code> Match files that start with any letter from 't' to 'y' - <code>ls /etc/[t-y]*</code>
!	Used with [ ] to negate a range	Match files that do NOT start with a 't' or 'y' - <code>ls /etc/[!ty]</code>

# WORKING WITH DIRECTORIES AND FILES

## **touch [filename]**

- Creates an empty file with the given file name
- Note: Linux filenames are at most 255 characters in length

## **rm [filename]**

- Deletes a file or directory

Option	What it does
-i	Interactive mode (ask before delete)
-r	Deletes a directory including contents

# WORKING WITH DIRECTORIES AND FILES

## `mkdir [directory]`

- Creates an empty directory with the given name

## `rmdir [directory]`

- Deletes a directory if empty

Option	What it does
-i	Interactive mode (ask before delete)



# WORKING WITH DIRECTORIES AND FILES

## **cp [source] [destination]**

- Copy file specified by source to destination

Option	What it does
-i	Interactive mode (ask to overwrite if existing)
-n	Do not overwrite
-r	Copy including folders

## **mv [source] [destination]**

- Move file specified by source to destination
- File retains name if destination filename is not specified
- Also has the `-i` and `-n` options

# GETTING HELP

## `man [command]`

- Opens the manual for a specified command
- Man pages describe what a command does and details of its options
- Typical sections of a man page:

Section	What's written
NAME	Provides the command name and a brief description
SYNOPSIS	Provides examples of the command is executed
DESCRIPTION	More detailed description of the command
OPTIONS	Lists options and description of how they are used
FILES	Lists files associated with the command if additional features can be configured
SEE ALSO	Provides where to find additional information about the command

# GETTING HELP

## man [command]

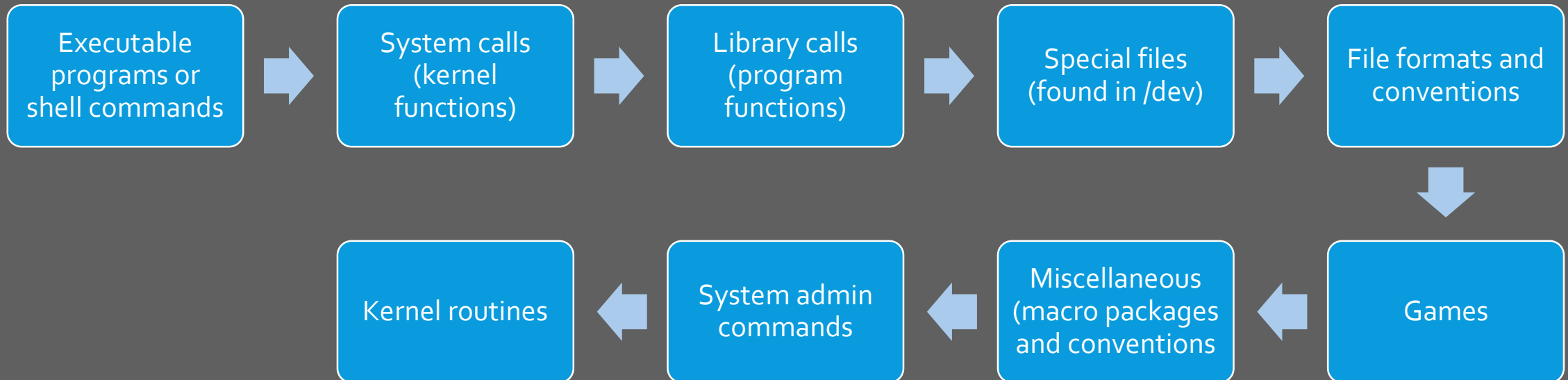
- To navigate man pages:

Key	What it does
f or Spacebar	Scrolls down
b	Scrolls up
/[word]	Searches for [word] in the man page
n	Find next match if searching
q	Quits the manual

# GETTING HELP

## `man [command]`

- Man pages are organized into sections
- Linux searches sections in order until it finds the first page that matches the command being looked up



# GETTING HELP

## man [command]

- Some man command options

Options	What it does
-k	Search for man pages that have words matching the given command
-f	Shows man pages for commands that match / partially match the string
-S [section #]	Open man page in specified section number Alternative command: man [section#] [command]

# UTILITIES

## **cat [filename]**

- Print contents of a file to standard output (usually the terminal)
- Cannot pause the output → Not practical for long files

## **more [filename] or less [filename]**

- Print contents of file to standard input with paging capability
- Navigation is the same as using man pages (space to scroll down, 'b' to scroll back, 'q' to quit)
- The 'less' command is not always found in Linux distributions, but has more features than 'more'
- In some Linux distributions, the 'more' command is mapped to the 'less' command

# UTILITIES

## **head [filename]**

- Print first few lines of the file (first 10 by default)

## **tail [filename]**

- Print last few lines of the file (last 10 by default)
- Note: Both commands allow specifying the number of lines to output using the **-n** option (e.g. head -n 5 file1.txt)

# UTILITIES

## grep [pattern] [filename]

- Filters file contents or command outputs to print only the lines that contain the specified pattern

Options	What it does
-n	Show line number
-c	Show only number of lines that were matched
--color	Highlight matched pattern
-i	Ignore case



# UTILITIES

## sort [filename]

- Rearrange the lines of files or input in either dictionary or numeric order based on the contents of one or more fields.
- By default, fields are assumed to be delimited by a space

Options	What it does
-r	Reverse sort
-t [char]	Set delimiting character
-k [number]	Sort by specified field number
-n	Numeric sort

# COMMAND LINE PIPES

- The *pipe* | character is used to send the output of a command as input to another command
- Often used to refine / filter results of a command
- Ex: `ls -al /etc | head`

# I/O REDIRECTION

- Allows command line information to be passed to different streams
  - STDIN – information entered by the user via the keyboard
  - STDOUT – normal output of the commands – usually to the terminal
- Redirection characters
  - < - allows STDIN data to be sourced from a file instead of a keyboard
  - > - allows STDOUT data to be outputted to a file instead of the terminal
  - >> - same as '>' but appends output if file exists instead of overwriting

# ADVANCED FILE COMMANDS

## **ln [target file] [link name]**

- Used to create alternative names linked to a file or directory
  - Hard link (default)
    - Changes metadata only to add a new file name for the same file
    - Not able to link across file systems
    - File is accessible as long as at least 1 hard link exists
  - Soft / Symbolic link (-s option)
    - creates a new file that contains a link to the target file or directory
    - Able to link across file systems
    - Easy to identify when listing using ls command
    - No longer usable if original file is deleted

# ALIASES

- Used to substitute a command for another
- Commonly used to map longer commands to shorter key sequences.
- When created, exists only for the shell where the alias was created and will be deleted once the system is rebooted unless added to the `.bashrc` file of the user home directory

- To create or an alias:

**alias [name]='[command sequence]'**

Ex: alias getpass='sudo more /etc/passwd'

- To delete an alias:

**unalias [name]**      Ex: unalias getpass

# SETTING SHELL CHARACTERISTICS

**set [-o / +o] [option name]**

- Shell-wide characteristics are dependent on flags that control how the shell functions
- Use `-o` to turn on, `+o` to turn off
- Example options:
  - `noclobber` - prevent files from being overwritten by redirection
  - `verbose` – print shell input lines as they are read

# BASIC SCRIPTING

- Shell scripts is a file of executable commands that has been stored in a text file.
- Commonly used to automate tasks or repetitive commands
- Can use any shell command and logic operators
- To create scripts, you'll need
  - Variables
  - Input commands
  - Operators
  - Conditional Statements
  - Loops

# SHELL VARIABLES

- Feature that allows storing of data
- Variables are given names and stored temporarily in memory
- Lost when a terminal window or shell is closed unless saved in `.bash_profile` of the user home folder
- In a BASH Shell, variables may be
  - Local
    - Associated with user tasks and available only for the shell session where it is created.
    - Usually lower case in convention
    - To create or change a variable:

**[Variable name] = [value]**    Ex: price=100

**[Variable name] = `[command]`**    Ex: directory=`pwd`

- To retrieve value:

**[\$[Variable name]]**    Ex: echo \$price



# SHELL VARIABLES

- In a BASH Shell, variables may be
  - Environment
    - Used by the shell in interpreting commands and performing tasks
    - Available to current and all subshells created
    - To create or change a variable:

**export [Variable name] = [value]**      Ex: export HISTSIZE=100

**export [existing variable name]**      Ex: export \$HISTSIZE

- To retrieve value:

**[\$Variable name]**      Ex: echo \$HISTSIZE

- To view existing environment variables:

**env**

# ACCEPTING INPUT

## `read [variablename]`

- Accepts a string from the keyboard and saves it to a variable
- Example: `read VAR1`

# SPECIAL CHARACTERS

- Parentheses ( )
  - Used to group commands together
  - Ex: `(pwd; ls -l) > out.txt`
- Double parentheses `(( ))`
  - Used for arithmetic operations
  - Ex: `echo $((1+1))`
- Braces `{ }`
  - Used to delimit variable names
  - Useful when trying to concatenate variable values with another string
  - Ex: `echo ${var1}_hello`
- Backslash `\`
  - Escape character used so that special characters are printed instead of interpreted by the shell
  - Ex: `echo \"$2`

# CONTROL STATEMENTS

- Allow multiple commands to be run at once or to run depending on success of a previous command.
- Semicolon (;)
  - Used to run commands consecutively and independent of each other
  - Ex: `cal 1 2016; cal 2 2016; cal 3 2016`
- Double Ampersand (&&)
  - Acts like a logical 'and'
  - Used to run the next command only if the previous one was run successfully
  - Ex: `rm file1 && echo 'Success!'`
- Double Pipe (||)
  - Acts like a logical 'or'
  - Skips command 2 if command 1 is successful and vice versa
  - Ex: `rm file1 || echo 'Delete failed'`

# COMPARATORS

**test [val1] [operator] [val2]**

- Outputs a 1 or a 0 exit code depending on results of the operation
- Operators:

Operator Type	Available Operators	Example
File	-e (exists)	test -e file1
String	=, !=, /<, />	test "a" = "a"
Arithmetic	-eq, -ne, -le, -lt, -ge, -gt	test \$var eq 1
Logic	-a, -o	test 1 -eq 1 -a 2 -eq 2

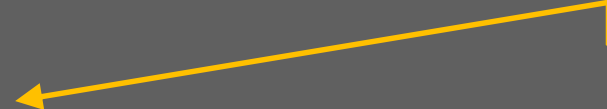
- Note:
  - Command can also use **[ expression ]** instead of "test" e.g. [ "a" = "a" ]
  - Each value is considered a command argument – spaces are important!

# BASIC SCRIPTING

- Basic script content:

```
#!/bin/sh
```

Means spawn a new BASH shell



```
# This is a comment
```

```
echo Hello World
```

```
# This is also a comment
```

- To run scripts: **`./[script path/name]`**
  - Make sure to set the 'x' flag using chmod

# CONDITIONALS – IF/ELSE

```
if command / condition
then
    do something
Elif command / condition
then
    do something else
else
    do something else
fi
```

Example:

```
read var1
if [ $var1 -eq 0 ]
then
    echo "Zero"
Elif [ $var1 -gt 1]
then
    echo "Positive"
else
    echo "Negative"
fi
```

# CONDITIONALS - CASE

```
case "variable" in
value1)
    do something
    ;;
value2)
    do something else
    ;;
*)
    do default action
esac
```

Example:

```
echo "Delete file1? (Y/N)"
read var1
case "$var1" in
Y|y)
    rm file1
    echo "file1 deleted."
    ;;
N|n)
    echo "Did not delete."
    ;;
*)
    echo "Invalid input."
esac
```



# LOOPS - WHILE

```
while condition
do
    do something
done
```

Example :

```
i=1
while [ i lt 6 ]
do
    echo "printing $i"
    i=$((i + 1))
done
```

# LOOPS - FOR

```
for var in list
do
    do something
done
```

Example 1:

```
for i in 1 2 3 4 5
do
    echo $i
done
```

Example 2:

```
list="str1 str2 str3 str4"
for i in $list
do
    echo $i
done
```