software carpentry

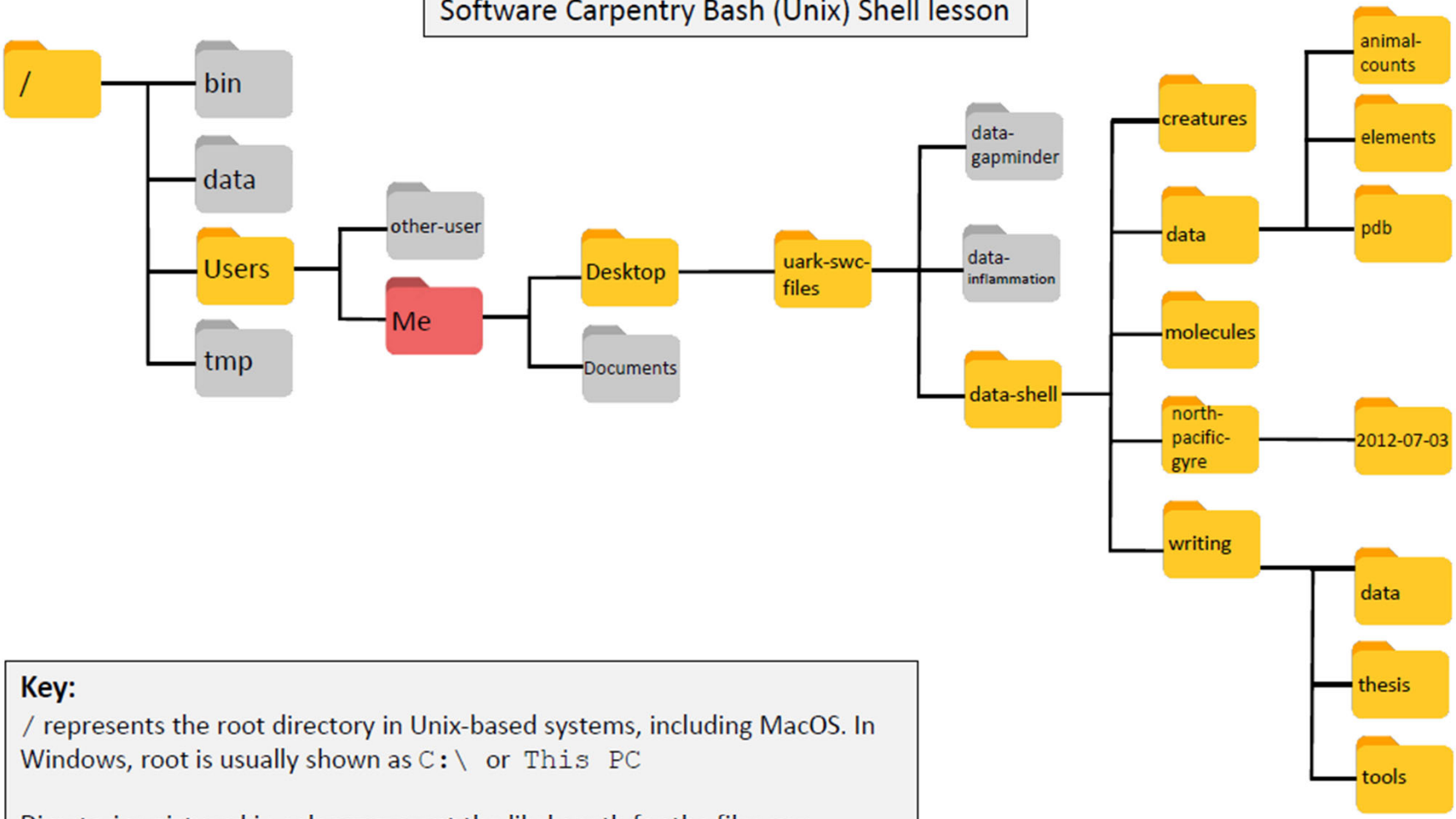
University of Arkansas

March 18-19, 2019

Shell/Bash

Directory Map

Software Carpentry Bash (Unix) Shell lesson



Key:

`/` represents the root directory in Unix-based systems, including MacOS. In Windows, root is usually shown as `C:\` or `This PC`

Directories pictured in color represent the likely path for the files you downloaded for the Shell lesson.

The red directory represents your likely default home directory.
This is the directory you land in when you type `cd` at the command prompt.

This map shows directories only. Your directories will also contain files.

Jamene Brooks-Kieffer
University of Kansas Libraries
https://github.com/kulibraries/swc-workshop-helps/blob/master/directory-map.pdf

HELP! 

Where's my prompt??!?!

Key one of these:

- Ctrl + C
- `q`

Key Points: navigating files & directories

- Information is stored in files, which are stored in directories (folders)
- Directories can also store other directories, which forms a directory tree
- ‘pwd’ prints the user's current working directory at the prompt
- ‘ls *path*’ prints a listing of a specific file or directory; ‘ls’ on its own lists the current working directory
- ‘cd *path*’ changes the current working directory
- Relative path: a location starting from the current location
- Absolute path: a location from the root of the file system
- ‘ .. ’ is the directory above the current one
- ‘ . ’ is the current directory

Names for files & directories

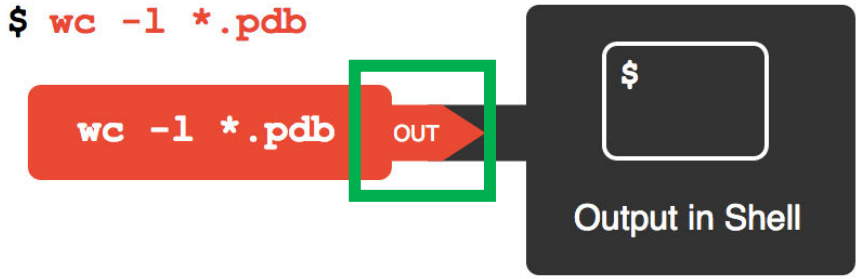
- Don't use spaces
  - Use - or _ instead
    - Bad: 2019 03 18 data readings
    - Good: 20190318_data-readings
- Don't begin the name with - (dash)
  - The dash is used for options in commands
    - Example: ls –F
- Don't use special characters
  - Stick with letters, numbers, ., -, _

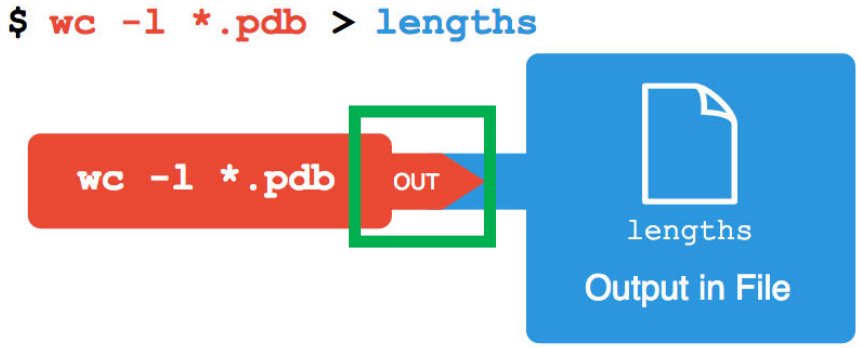
Key Points: working w/files & directories

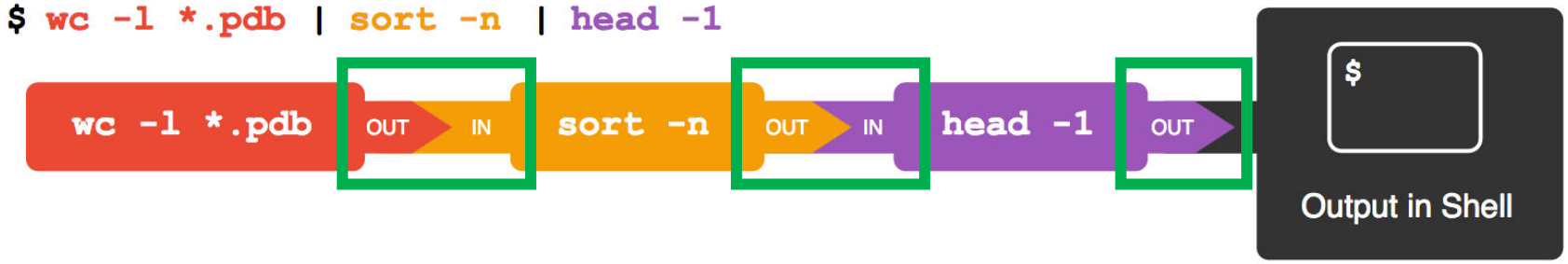
- 'cp *old new*' copies a file
- 'mkdir *path*' creates a new directory
- 'mv *old new*' moves (renames) a file or directory
- 'rm *path*' removes (deletes) a file
- '*' is a wildcard – it matches zero or more characters in a filename, so '*.txt' matches all files ending in '.txt'
- '?' is a wildcard – it matches any single character in a filename, so '?.txt' matches 'a.txt' but not 'any.txt'
- The shell does not have a trash bin: once something is deleted, it's really gone
- Most files' names are 'something.extension'
    - The extension isn't required, and doesn't guarantee anything, but is normally used to indicate the type of data in the file
- Depending on the type of work you do, you may need a more powerful text editor than Nano

Processes using the shell

Processes move through standard input and standard output







Key Points: pipes & filters

- 'cat *filename*' prints the contents of files on the terminal
- 'head' displays the first 10 lines; 'tail' displays the last 10 lines
- 'sort' sorts its inputs
- 'wc' (word count) counts lines, words, and characters in its inputs
- '*command* > *file*' **redirects** a command's output to a file (overwriting any existing content)
- '*command* >> *file*' **appends** a command's output to a file
- 'echo' prints text in the terminal
- ' < ' redirects input to a command
- '*first* | *second*' is a pipeline: the output of the first command is used as the input to the second

Key Points: loops

- A `for` loop repeats commands once for everything in a list
- Every `for` loop needs a variable to refer to the thing it is currently operating on
- Use `$name` to expand a variable (i.e., get its value)
- Do not use spaces, quotes, or wildcard characters in filenames
- Give files consistent names that are easy to match with wildcard patterns to make it easy to select them for looping
- Use the up-arrow key to scroll up through previous commands to edit and repeat them
- Use `Ctrl+R` to search through the previously entered commands
- Use `history` to display recent commands, and `!number` to repeat a command by number

How to write a loop

Type

$ for *filename* in *directory/file*

> do

> *command $filename*

> *command* *may have multiple commands

> done

Press return

$ *results printed to screen*

Key Points: scripts

- Save commands in files (usually called shell scripts) for re-use
- `bash *filename*` runs the commands saved in a file
- `$@` refers to all of a shell script's command-line arguments
- `$1`, `$2`, etc., refer to the first command-line argument, the second command-line argument, etc.
- Place variables in quotes if the values might have spaces in them
- Letting users decide what files to process is more flexible and more consistent with built-in Unix commands