LINUX

AN INTRODUCTION TO THE BASH SHELL

LINUX

AN INTRODUCTION TO THE BASH SHELL

THE LINUX SHELL IS ALSO KNOWN AS THE COMMAND-LINE

LINUX

AN INTRODUCTION TO THE BASH SHELL

THE LINUX SHELL IS ALSO KNOWN AS THE COMMAND-LINE

- (1999) in the beginning ... was the command line
- (2010) a taxonomy of data-science
- (2019) data science at the command line

From "A taxonomy of Data Science":

From "A taxonomy of Data Science":

> POINTING AND CLICKING DOES NOT SCALE.

From "A taxonomy of Data Science":

> **POINTING AND CLICKING DOES NOT SCALE.**

> *Part of the skillset of a data scientist is knowing how to obtain a sufficient corpus of usable data... At a minimum, a data scientist should know how to do this from the command line, e.g., in a UN*X environment.*

From "A taxonomy of Data Science":

> **POINTING AND CLICKING DOES NOT SCALE.**

> *Part of the skillset of a data scientist is knowing how to obtain a sufficient corpus of usable data... At a minimum, a data scientist should know how to do this from the command line, e.g., in a UN*X environment.*

> *(mentioned) shell-scripting, python/perl, sed, awk,grep, less, head, cut, ...*

From a review of "Data Science at the Command Line:

From a review of "Data Science at the Command Line:

> *The Unix philosophy of simple tools, each doing one job well, then cleverly piped together, is embodied by the command line. Jeroen expertly discusses how to bring that philosophy into your work in data science, illustrating how the command line is not only the world of file input/output, but also the world of data manipulation, exploration, and even modeling.*

From a review of "Data Science at the Command Line:

> *The Unix philosophy of simple tools, each doing one job well, then cleverly piped together, is embodied by the command line. Jeroen expertly discusses how to bring that philosophy into your work in data science, illustrating how the command line is not only the world of file input/output, but also the world of data manipulation, exploration, and even modeling.*

Chris H. Wiggins,
Associate Professor, Dept of Applied Physics and Applied Mathematics, Columbia University,
Chief Data Scientist, The New York Times

USER INTERFACES

There are two ways we usually interact with a computer running the Linux operating system:

USER INTERFACES

There are two ways we usually interact with a computer running the Linux operating system:

1. **COMMAND-LINE INTERFACE (CLI OR SHELL)**

USER INTERFACES

There are two ways we usually interact with a computer running the Linux operating system:

1. **COMMAND-LINE INTERFACE (CLI OR SHELL)**

    - Thompson shell (1971)

USER INTERFACES

There are two ways we usually interact with a computer running the Linux operating system:

1. **COMMAND-LINE INTERFACE (CLI OR SHELL)**
    - Thompson shell (1971)
    - Bourne shell (1977)

USER INTERFACES

There are two ways we usually interact with a computer running the Linux operating system:

1. **COMMAND-LINE INTERFACE (CLI OR SHELL)**
    - Thompson shell (1971)
    - Bourne shell (1977)
    - bash shell, "the Bourne-again shell" (1989)

USER INTERFACES

There are two ways we usually interact with a computer running the Linux operating system:

1. **COMMAND-LINE INTERFACE (CLI OR SHELL)**
    - Thompson shell (1971)
    - Bourne shell (1977)
    - bash shell, "the Bourne-again shell" (1989)
    - and a lot of others

USER INTERFACES

There are two ways we usually interact with a computer running the Linux operating system:

1. **COMMAND-LINE INTERFACE (CLI OR SHELL)**
    - Thompson shell (1971)
    - Bourne shell (1977)
    - bash shell, "the Bourne-again shell" (1989)
    - and a lot of others
2. **GRAPHICAL USER INTERFACE (GUI)**

    1987

THE SHELL

The shell is your text interface with the linux kernel. It provides

THE SHELL

The shell is your text interface with the linux kernel. It provides

- interactive use

THE SHELL

The shell is your text interface with the linux kernel. It provides

- interactive use
- customization of your linux session

THE SHELL

The shell is your text interface with the linux kernel. It provides

- interactive use
- customization of your linux session
- programming constructs - shell commands used to create scripts

THE SHELL

The shell is your text interface with the linux kernel. It provides

- interactive use
- customization of your linux session
- programming constructs - shell commands used to create scripts

The default shell on many linux distributions and MacOS is the bash shell.
The Windows Subsystem for Linux also provides the bash shell.

THE TERMINAL

You need a terminal window open to access the shell.

- Try "Ctrl-Alt-t" in linux or MacOS.
- *In that window, you are "at the command-line", or "at the shell".*

We'll be spending all our time at the shell.

THE SHELL PROMPT

The bash shell prompt is typically

```
$
```

If you see that prompt, type *what follows* at the command-line on your terminal.
Then press enter.

```
```

THE SHELL PROMPT

The bash shell prompt is typically

```
$
```

If you see that prompt, type *what follows* at the command-line on your terminal.
Then press enter.

```
$ ls
```

THE SHELL PROMPT

The bash shell prompt is typically

```
$
```

If you see that prompt, type *what follows* at the command-line on your terminal.
Then press enter.

```
$ ls
```

The character "#" denotes the beginning of a comment - don't type it or what follows.

THE SHELL PROMPT

The bash shell prompt is typically

```
$
```

If you see that prompt, type *what follows* at the command-line on your terminal.
Then press enter.

```
$ ls
```

The character "#" denotes the beginning of a comment - don't type it or what follows.

```
$ pwd  # print present working directory
```

THE SHELL PROMPT, CONTINUED:

THE SHELL PROMPT, CONTINUED:

The superuser prompt is

```
#
```

THE SHELL - INTERACTIVE USE

```
$ whoami                # whoever you logged in as
$ w                     # who's logged on, what are they doing?
$ cd                    # move to home directory
$ pwd                   # show full path of present working directory
$ ls                    # list files in the current directory.
$ history               # display shell history
$ top                   # display all running processes
```

SHELL COMMANDS WITH OPTIONS AND ARGUMENTS, AND PIPES

- Options start with "-", arguments come last.
- A pipe "|" takes the output of the previous command as input to the next command.

```
$ id                          # 1) print your user/group information
$ top -u $USER                # 2) show all jobs you are currently running
$ ps ux                       # 3) another way of seeing your running jobs
$ ls /var/log                 # 4) list files in the directory /var/log
$ ls -lt /var/log/*.log       # 5) list ".log" files in the given directory
$ history 20                  # 6) display last 20 entries in shell history
$ history| tail -n 10         # 7) from the output of the history command,
                              #    show the last 10 lines
$ history| head -n 4          # 8) from the output of the history command,
                              #    show the first 4 lines
$ man command-of-interest # 9) help on "command-of-interest"
```

BASIC NAVIGATION

MOVING AROUND

```
$ cd                       # 1) change directory to "home"
$ mkdir temp_dir           # 2) make directory (folder)
$ mkdir temp_dir/a/b       # 3) make folder b within new folder a (fails)
$ mkdir -p temp_dir/a/b    # 4) make folder b within new folder a (succeeds
$ man mkdir                # 5) how does that command work?
$ cd temp_dir/a/b          # 6) change directory
$ pwd                      # 7) print present working directory
$ cd ..                    # 8) navigate up one directory
$ pwd; ls                  # 9) where are we? / what's here?
$ cd -                     #10) return to previous directory
$ cd ../../..              #11) navigate up three directories
$ ls -lRt temp_dir         #12) list all files recursively from temp_dir
$ cd /tmp                  #13) move to /tmp directory (temporary storage)
```

Paths contain "/".

Absolute paths start with "/".

BASIC FILE-HANDLING

WORKING WITH FILES

```
$ cd                         # 1) move to home directory
$ nano data_file             # 2) write content into file
```

```
Enter the lines:
first row
second row
third row
fourth row
third row
Write out with Crtl-O, Ctrl-X.
```

```
$ cp data_file data_file_$(date +%F)
                             # 3) make a copy of data_file, appendi
$ ls                         # 4) see if it worked
$ wc -c data_file            # 5) character count of that file
$ wc -w !$                   # 5) word-count of that file
$ wc -l !$                   # 6) line-count of that file
```

SHELL UTILITIES

UTILITIES, INPUT, OUTPUT, PIPES

You can use the output of commands as input of other commands through Unix pipes(|):

```
$ cat data_file|sort > data_file2 # 1) sort lines and write to another fi
$ cat data_file2                  # 2) see result
$ cat data_file|sort | uniq       # 3) only show unique lines of file
$ cat data_file|sort | uniq -d    # 4) only show duplicate lines of file
```

HELP

There are two types of shell utilities

1) **shell builtins**

basic tools documented with the command "help":

```
$ help          # see them all
$ jobs          # all jobs running in this shell
$ help jobs
ctrl-z          # put current foreground job in background
$ fg %1         # resume suspended job #1 in the foreground
```

2) external shell commands (programs built from built ins)

documented in the manual pages:

```
$ ls /usr/bin              # see many of them
$ whereis python           # where are all the available python executabl
$ which python             # where is my python executable?
$ whatis python            # one-line description from man-page
$ man python               # full man-page for help on python
```

THE MANUAL IS YOUR FRIEND

when you know what you're looking for

... AND SO IS "APROPOS"

when you don't

```
$ man apropos
apropos - search the manual page names and descriptions

DESCRIPTION
 Each manual page has a short description available within it.
 apropos searches the descriptions for instances of keyword.
```

Use apropos to find the command you need.

```
$ apropos permissions
```

THE SHELL - CUSTOMIZING YOUR LINUX ENVIRONMENT

The shell by default sets the values of many variables.
Their names are always upper-case.
Their values are accessed by putting a $ in front.

```
$ env                # see your environment variables
$ set                # local and environment variables
$ cd                 # same as "cd $HOME"
$ echo $USER         # same as "whoami"
$ echo $PATH         # where your system looks for your commands *
$ cat .bashrc        # where you can configure your shell (linux)
$ cat .bash_profile # where you can configure your shell (macOS)
```

The history command tells you what commands you've issued in the past:

```
$ history | tail -5  # for the last 5 commands used
```

The history command tells you what commands you've issued in the past:

```
$ history | tail -5  # for the last 5 commands used
```

Configure the history command with:

```
$ export HISTTIMEFORMAT="%F %T "
```

The history command tells you what commands you've issued in the past:

```
$ history | tail -5  # for the last 5 commands used
```

Configure the history command with:

```
$ export HISTTIMEFORMAT="%F %T "
```

then do some stuff ..., and check your history again

```
$ history| tail -5
```

The history command tells you what commands you've issued in the past:

```
$ history | tail -5  # for the last 5 commands used
```

Configure the history command with:

```
$ export HISTTIMEFORMAT="%F %T "
```

then do some stuff ..., and check your history again

```
$ history| tail -5
```

You get a timestamp associated with each command. Very handy!

The format for history you will then see is:

```
$ history |tail -5
2019-07-09 16:24:06:  env
2019-07-09 16:24:11:  set
2019-07-09 16:24:18:  set|grep USER
2019-07-09 16:24:23:  vi index.html
2019-07-09 16:25:02:  history 10
```

ASIDE

To make this configuration permanent, add the command

```
$ export HISTTIMEFORMAT="%F %T "
```

to your bash configuration file

- ~/.bashrc (linux)
- ~/.bash_profile (MacOS) , and source that file to

activate the setting:

```
$ source ~/.bashrc #or source ~/.bash_profile
```

ASIDE CONTINUED

The settings in this configuration file are applied to every terminal shell you open. Add more settings as you like.

Many utilities/programs you install have their own configuration files. They are usually *hidden* in your home-directory - they start with a dot, and aren't listed by default. To see them:

```
$ ls -lat      # everything
$ ls -lat .*   #just hidden (dot) files
```

THE SHELL - PROGRAMMING WITH SHELL SCRIPTS

- store convenient executable functions in ~/bin
- ~/bin is in your PATH
- executable files stored in your path can be run by just typing their name.

```
$ chmod u+x filename.sh # make it executable
$ filename.sh           # run it if on your $PATH
$ ./filename.sh         # if not on your $PATH
```

SHELL SCRIPT EXAMPLE

```
$ nano make_some_files.sh
```

Enter the lines:

```
  #!/bin/bash
for dir_number in `seq 1 10`;
do mkdir dir_$dir_number;
    for  file_number in `seq 1 20`;
    do
        touch dir_$dir_number/file_${file_number}_$(date +%F);
    done;
done;
```

```
$ ls                                # 1) before
$ chmod u+x make_some_files.sh      # 2) make it executable
$ ./make_some_files.sh              # 3) execute it
$ ls -Rt                            # 4) see what's changed
```

ENOUGH!!

THANK YOU.

2019 SOFTWARE CARPENTRY

- In mid-August, University of Auckland will be hosting another Software Carprentry event.
- You're encouraged to attend
- the perfect opportunity to increase your software skills
- Attendance is free, but you must enrol.