8.1 The Command Line

8.1.1 Introduction to the Command Line Interface

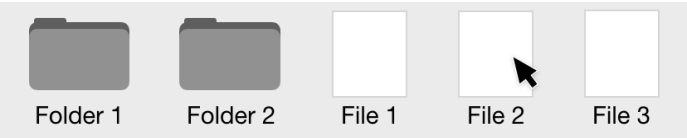
Computer Interfaces

Most of your prior experiences using a computer have likely been via a Graphical User Interface (GUI) (pronounced "gooey"). GUIs allow interaction between user and computer via a mouse or track pad. Actions are performed by moving a cursor around the screen and clicking on graphical icons (like the save button in word processing applications). Clicking on an icon tells the computer to execute a **command**. A command is an instruction given to a computer to perform a specific action, for example, "save a file", "open a file", "go back a page", etc. Interacting with a computer via a GUI can be broken down into three steps:

move cursor > click icon > computer executes command

For example, to open a file in the file browser depicted below one would click the file icon:

place cursor over file > click file icon > computer opens file



Before the invention of GUIs, the only way to interact with a computer was via the Command Line Interface (CLI). This is an entirely text-based system for interacting with a computer. Instead of a mouse, commands must be provided via keyboard. Interacting with a computer via a CLI can be broken down into three steps:

type command > press enter > computer executes command

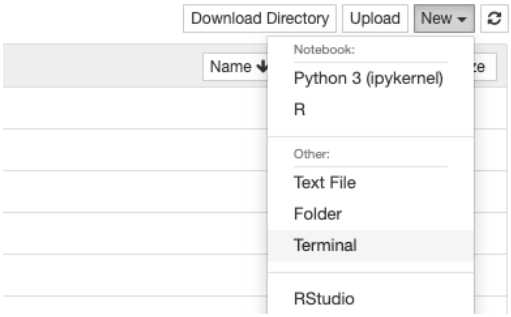
These commands are read and executed by a **command interpreter** also called the **shell**. You may be most familiar with command line interfaces as they are seen in movies or on the news, with a black background and green text with lines of code. Below is a depiction of the MS-DOS CLI from 1981

```
MS-DOS version 1.25
Copyright 1981,82 Microsoft, Inc.

The CDP Personal Computer DOS
Version 2.11 (C)Copyright Columbia Data Products, Inc. 1982, 1983
Current date is Tue  1-01-1980
Enter new date:
Current time is: 0:00:06.15
Enter new time:

A:_
```

(**Code taken from:** https://betawiki.net/wiki/MS-DOS_1.25#/media/File:MS-DOS-1.25-Interface.png)

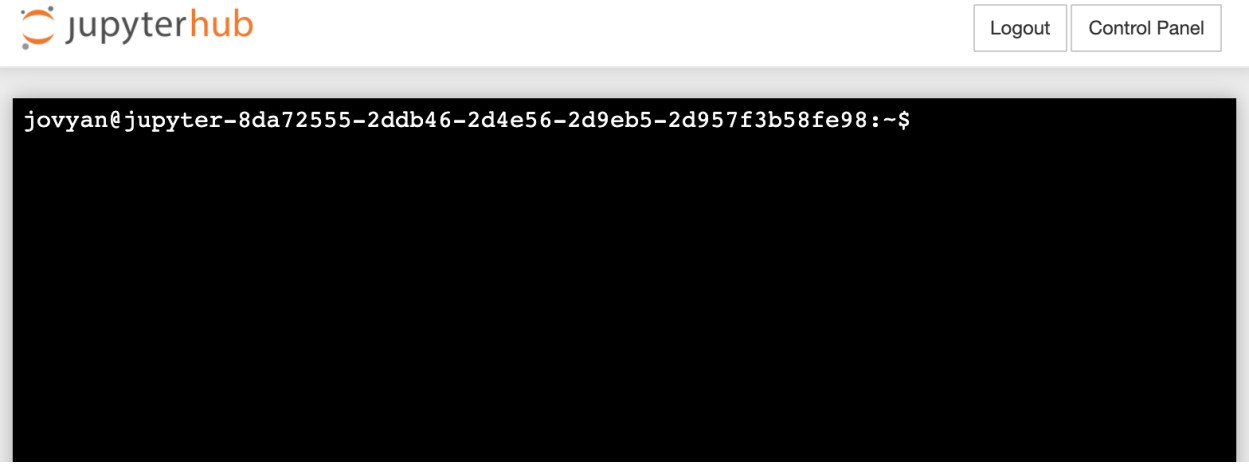
Terminal

An **operating system (OS)** is the software on a computer that manages & monitors computer hardware, transmits information between the user and the hardware, and manages computer applications. The OS also handles the file system. The computer you are working on now has an operating system, the most common being Microsoft Windows, macOS, and Linux operating systems.

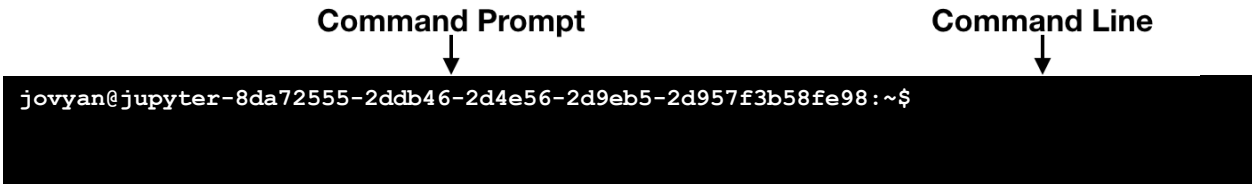


Terminal is a command line interface application on macOS and Linux operating systems, including JupyterHub. JupyterHub uses the Linux operating system Ubuntu. To open a new Terminal session in Jupyter Hub, click New > Terminal on the upper right side of the JupyterHub landing page.

When a new Terminal session is opened in the command line, a new tab will open in the internet browser, as illustrated below. The black box with white text is the command line interface. In weeks 8-11 it is suggested that you open Terminal in JupyterHub and perform the commands in the lecture supplements in order to learn to interact with the computer using the command line. Note that the commands taught in this section may not work the same way on other operating systems or command line interface applications.



The visible white text is called the **command prompt** and the rest of the line following it is called the **command line**.



The command prompt is the text at the start of each line in the command line interface and contains multiple components. The format of the command prompt differs between command line interface applications and command interpreters. In Terminal, the command

prompt specifies the name of the user, the name of the computer, and the **present working directory (PWD)** in the format:

`username@computer:PWD$`

For example, in the above prompt the user is `jovyan`, the computer is `jupyter-8da72555-2ddb46-2d4e56-2d9eb5-2d957f3b58fe98`, and the current directory is `~`.

As a reminder, the present working directory (PWD) is the directory in which the user is currently working. The tilde symbol (~) is used to represent the user's **home directory**. The home directory is the directory that holds a user's personal files. See 9.1.2 for further details.

Note that, in the lecture slides and supplements, the full command prompt will not be shown. To conserve space, only the letter j will be used as a stand-in for the user and computer name. The PWD will still be shown in its entirety.

For example, instead of:

```
jovyan@jupyter-8da72555-2ddb46-2d4e56-2d9eb5-2d957f3b58fe98:~$
```

Only the following will be shown.

```
j:~$
```

A command is executed in the Terminal by typing the command in the command line after the command prompt and pressing ENTER.

type command > press ENTER > command interpreter (shell) reads & executes command

In Terminal, the name of the command interpreter (or shell) is called **bash**. Two examples of commands recognized by bash are `whoami` and `date`. The `whoami` command returns the name of the user and the `date` command returns the current date and time.

To run the `whoami` command, type `whoami` after the command prompt:

```
j:~$ whoami
```

When you press enter, you will see the following:

```
j:~$ whoami
jovyan
j:~$
```

The line following the command contains the output of the command. Immediately following the output is a new command prompt. A new command prompt appears each time a command is executed to facilitate typing and running the next command. Open Terminal in JupyterHub and execute this command to see how it works.

To run the date command, it can be typed after the new command prompt:

```
j:~$ whoami
jovyan
j:~$ date
```

When you press enter, the output is returned along with a new command prompt:

```
j:~$ whoami
jovyan
j:~$ date
Tue 14 Dec 2021 03:54:46 PM EST
j:~$
```

In the rest of this lecture supplement and all following lecture supplements, instead of showing the full terminal after each command, the code will be presented as follows:

```
j:~$ whoami
jovyan
j:~$ date
Tue 14 Dec 2021 03:54:46 PM EST
j:~$
```

Command Line Interface Tips

If you do not already have it open, open Terminal in JupyterHub and execute the whoami and date commands as in the above section.

In word processing applications such as Microsoft Word or Pages there is a text cursor, a visual blinking vertical line that signals where the letters you type will appear. To move your text cursor to edit text in a different part of a document, you can click in another place in the document, and it will move to the new location.

In a command line interface, the mouse has no role, thus you cannot move to a different location in the command line using your mouse. Instead, you must use the left and right arrows on the keyboard to change the location in which you are typing. The text cursor in Terminal appears as a white block.

Type the `whoami` command again and try to use your mouse to move the cursor to the start of the command. It will not work. Instead, use the left arrow key to move the cursor to the start of the line.

You can also use keyboard shortcuts for moving the cursor:

- Move the text cursor to the beginning of the command line
    - Microsoft computers: Ctrl + Home
    - Mac computers: Ctrl + A
- Move the text cursor to the end of the command line
    - Microsoft computers: Ctrl + End
    - Mac computers: Ctrl + E

Press enter to execute the `whoami` command you have typed. Now, type "whoa" into the command line at the new command prompt:

```
j:~$ whoa
```

Press the tab button on your keyboard and the command will automatically be completed as `whoami`. This is called tab completion and works the same way as it does in R (see supplement 1.3). If there is only one possible command that starts with the letters that have already been typed, it will fill in the rest of the command. In this case, only the `whoami` command starts with "whoa" so the command was completed.

Press enter to bring up a new command prompt. Now type "wh" into the command line at the new command prompt and press the tab button. This will output all the commands the start with "wh" and create a new command prompt that already contains the letters "wh":

```
j:~$ wh
whatis          whereis         while           whoami
wheel           which           who             whoopsie-preferences
j:~$ wh
```

Complete typing the `whoami` command and press enter. Now type the command `date` and press enter, and then type the command `cal` and press enter:

```
j:~$ whoami
jovyan
j:~$ date
Tue 14 Dec 2021 04:17:04 PM EST
j:~$ cal
   December 2021
Su Mo Tu We Th Fr Sa
           1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
j:~$
```

The `cal` command outputs a calendar of the current month. Note that commands can return multiple lines of output.

In the new empty command line after the command prompt press the up arrow once. The previous command executed in the command line (`cal`) will appear. If you press the up arrow again it will show the command executed prior to that (`date`), etc. Now if you press the down button, you can travel back through the executed commands to the empty command prompt.

Note that tab completion will also work with file names.

8.1.2 Navigating the Directory Structure

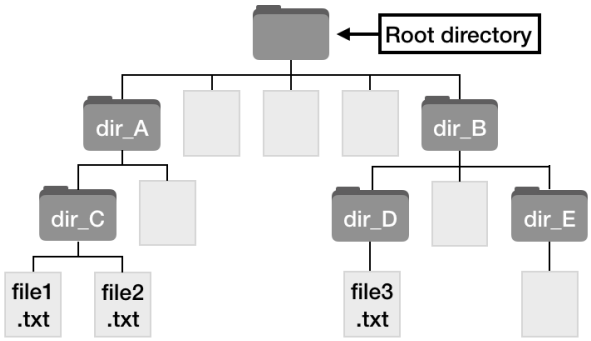
Directory Structure

In 5.2 the directory structure was introduced. Key definitions are summarized here.

Directory: a file organizing structure (often called a "folder").

Directory structure: the file organization system of an OS, usually a hierarchical tree structure.

The directory structure illustrated below has 4 levels. Directories are represented by dark gray folder icons and files are displayed by light gray rectangular file icons. Black lines between files and directories denote which files are contained within which directories. For example, `file1.txt` and `file2.txt` are in `dir_C`, and `dir_C` is a directory in `dir_A`.



Parent directory: directory in which a file or another directory is contained. Every file or directory has exactly one parent directory.

- The parent directory of `dir_C` is `dir_A`
- The parent directory of `file1.txt` is `dir_C`

Child directory: directory contained within another directory. A directory can have many child directories.

- `dir_A` has one child directory: `dir_C`
- `dir_B` has two child directories: `dir_D` and `dir_E`

Root directory: directory that contains all other files and directories. This is the only directory that has no parent directory. The root directory is also the only directory that has no name.

Path: the location of a file or directory in a directory structure. In a path each directory in the route to the location is named and tree levels are separated by forward slashes: `/`

Absolute path: file or directory path starting from the root directory. Note that as the root directory has no name all absolute paths start with a forward slash.

- The absolute path of the file `file2.txt` is: `/dir_A/dir_C/file2.txt`
- The absolute path of the directory `dir_D` is: `/dir_B/dir_D`

Present working directory (PWD): the directory in which you (the user) are currently working.

Relative path: file or directory path starting from the present working directory. Parent directories (up one level) are referenced with two periods: `..`

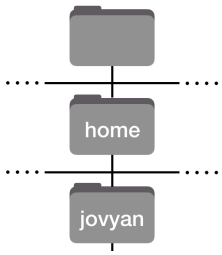
- If the PWD is `dir_B`, the relative path to the file `file3.txt` is `dir_D/file3.txt`
- If the PWD is `dir_E`, the relative path to the file `file3.txt` is `../dir_D/file3.txt`
- If the PWD is `dir_C`, the relative path to the file `file3.txt` is `../../dir_B/dir_D/file3.txt`

The pwd & ls Commands

The `pwd` command returns the absolute path of the present working directory. Like the `whoami` and `date` commands `pwd` is typed into the command line, enter is pressed and the output is returned on the following line:

```
j:~$ pwd
/home/jovyan
j:~$
```

Based on the output, the home directory (directory that holds a user's personal files) is called `jovyan` (the name of the user). The `jovyan` directory has a parent directory called `home` which is in the root directory. The structure is depicted to the right.



The `ls` command returns a list of files and directories. If the `ls` command is run on its own it returns a list of the files and directories in the PWD:

```
j:~$ ls
Week.1  Week.2  Week.3  Week.4  Week.5  Week.6  Week.7  Week.8
j:~$
```

The ls function can also be executed with **arguments**. In the command line an argument is data (often a file or directory) provided to a command. The syntax or usage of a command with arguments is shown as follows:

```
ls [file]...
```

`[file]...` indicates that the path of one or more files or directories can be provided to the `ls` command. Square brackets and lower-case letters indicate the argument is optional, and the ellipsis indicates that multiple arguments of the same type can be provided.

For example, if the PWD is ~ the items in the `Week.7` directory can be listed by providing the relative path to `Week.7` after the `ls` command.

```
j:~$ ls Week.7
7.1.Function.Creation 7.2.Data.Visualization Assignment.7.Rmd
j:~$
```

Navigating Directory Levels

The `cd` command changes the PWD.

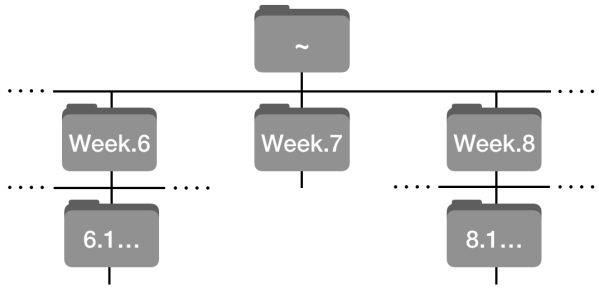
```
cd [dir]
```

`dir` is the absolute or relative path to a directory. In the home directory (PWD) is the `Week.8` directory which contains the directory `8.1.The.Command.Line`. To change the PWD to `8.1.The.Command.Line`:

```
j:~$ cd Week.8/8.1.The.Command.Line
j:~/Week.8/8.1.The.Command.Line$
```

The `cd` command produces no output; however, the command prompt will change. The part of the command prompt between the colon and the dollar sign contains the path of the PWD from the home directory (~).

In the visualization to the right, part of the home directory structure is displayed. The home directory has child directories `Week.6`, `Week.7`, and `Week.8`. In the `Week.6` directory is the directory `6.1.For.Loops`. To change the PWD to that directory:




```
j:~/Week.8/8.1.The.Command.Line$ cd ../../Week.6/6.1.For.Loops
j:~/Week.6/6.1.For.Loops$
```

Note again that the command prompt has changed to reflect the PWD. From this directory, the contents of other directories can be listed. To list the contents of the home directory, which is to levels above `6.1.For.Loops` the following command can be used:

```
j:~/Week.6/6.1.For.Loops$ ls ../..
Week.1  Week.2  Week.3  Week.4  Week.5  Week.6  Week.7  Week.8
j:~/Week.6/6.1.For.Loops$
```

Or, the contents of the `Week.7` directory can be listed:

```
j:~/Week.6/6.1.For.Loops$ ls ../../Week.7
7.1.Function.Creation  7.2.Data.Visualization  Assignment.7.Rmd
j:~/Week.6/6.1.For.Loops$
```

The contents of any directory can be listed from PWD; the only difference is the path you provide to get there.

The `cd` command does not require an argument. If run on its own it will return to the home directory:

```
j:~/Week.6/6.1.For.Loops$ cd
j:~$ pwd
/home/jovyan
j:~$
```

8.1.3 Modifying the Directory Structure

Creating Directories

A new directory can be made using the `mkdir` command.

```
mkdir DIRECTORY...
```

`DIRECTORY` is the absolute or relative path to a directory to create. In the commands below, the directory is changed to `8.1.The.Command.Line`, the contents are listed, and a new directory called `test_dir` is created.

```
j:~$ cd Week.8/8.1.The.Command.Line
j:~/Week.8/8.1.The.Command.Line$ ls
Supplement.8.1.pdf   testfile.txt   Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$ mkdir test_dir
j:~/Week.8/8.1.The.Command.Line$ ls
Supplement.8.1.pdf   testdir  testfile.txt   Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$
```

There was no output from the `mkdir` command, however when `ls` is run again a new item is in the directory called `test_dir`. Listing the contents for the new directory show that it is empty.

```
j:~/Week.8/8.1.The.Command.Line$ ls test_dir
j:~/Week.8/8.1.The.Command.Line$
```

Note that there should never be spaces in any directory or file names.

If the path provided for the new directory already exists, an error will output and nothing will change.

```
j:~/Week.8/8.1.The.Command.Line$ mkdir test_dir
mkdir: cannot create directory ‘test_dir’: File exists
j:~/Week.8/8.1.The.Command.Line$
```

Copying & Moving Files & Directories

The `cp` command copies files and directories to a new location.

```
cp SOURCE DEST
```

`SOURCE` is the absolute or relative path to the file or directory to copy and `DEST` is the absolute or relative path to the new location. For example, below the file `testfile.txt` is copied into `test_dir`:

```
j:~/Week.8/8.1.The.Command.Line$ ls
Supplement.8.1.pdf   test_dir  testfile.txt   Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$ ls test_dir
j:~/Week.8/8.1.The.Command.Line$ cp testfile.txt test_dir
j:~/Week.8/8.1.The.Command.Line$
```

There was no output from the `cp` command. Checking the contents of `test_dir` reveals that `testfile.txt` is now in the `test_dir` directory, but is also still in the `8.1.The.Command.Line` directory.

```
j:~/Week.8/8.1.The.Command.Line$ ls test_dir
testfile.txt
j:~/Week.8/8.1.The.Command.Line$ ls
Supplement.8.1.pdf   test_dir  testfile.txt   Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$
```

Note that the file `testfile.txt` in `test_dir` will contain the exact same contents as `testfile.txt`, the name is the same and the contents have not changed, however the files are in different locations.

Another copy a file can also be made with a new name. For example, to create a copy of the file `testfile.txt` in the same directory, but with the name `filecopy.txt` can be made with the following command:

```
j:~/Week.8/8.1.The.Command.Line$ ls
Supplement.8.1.pdf   test_dir  testfile.txt   Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$ cp testfile.txt filecopy.txt
j:~/Week.8/8.1.The.Command.Line$ ls
filecopy.txt  Supplement.8.1.pdf   test_dir  testfile.txt   Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$
```

Note that `filecopy.txt` file will contain the exact same contents as `testfile.txt`, the name is different, but the contents have not changed.

The `mv` command moves files and directories to a new location.

```
mv SOURCE DEST
```

SOURCE is the absolute or relative path to the file or directory to copy and DEST is the absolute or relative path to the new location. The mv command differs from the cp command in that the source file or directory will no longer exist after being moved but will still exist if it is copied. For example, below the file filecopy.txt is moved into test_dir:

```
j:~/Week.8/8.1.The.Command.Line$ ls
filecopy.txt  Supplement.8.1.pdf   test_dir  testfile.txt   Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$ ls test_dir
testfile.txt
j:~/Week.8/8.1.The.Command.Line$ mv filecopy.txt test_dir
j:~/Week.8/8.1.The.Command.Line$ ls
Supplement.8.1.pdf   test_dir  testfile.txt   Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$ ls test_dir
filecopy.txt  testfile.txt
j:~/Week.8/8.1.The.Command.Line$
```

After the file filecopy.txt is moved into test_dir it not is longer in the 8.1.The.Command.Line directory, but it is in the test_dir directory.

To rename a file it can be moved into a file with a new name. For example, to change the name of the file testfile.txt to newname.txt, testfile.txt can be moved to newname.txt:

```
j:~/Week.8/8.1.The.Command.Line$ ls
Supplement.8.1.pdf   test_dir  testfile.txt   Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$ mv testfile.txt newname.txt
j:~/Week.8/8.1.The.Command.Line$ ls
newname.txt  Supplement.8.1.pdf   test_dir  Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$
```

The newname.txt file will contain the contents of testfile.txt, only the name has changed.

Deleting Files & Directories

Commands can have **options** as well as arguments. An option is a letter or word that determines the behaviour of a command. Options are often also called **flags**, and they are always optional. The usage syntax for options is:

```
command OPTIONS ARGUMENTS
```

The rm command removes files and directories.

```
rm [OPTION]... [FILE]...
```

FILE is the absolute or relative path to the file or directory to remove. There are a number of options for the remove command, include the -r option which tells the rm command to remove recursively. The -r option must always be used when a removing a directory as it

tells the command to remove not only the directory but all the files and child directories inside the directory.

To remove the file `newname.txt` in the `8.1.The.Command.Line` directory, the following command can be run:

```
j:~/Week.8/8.1.The.Command.Line$ ls
newname.txt  Supplement.8.1.pdf   test_dir  Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$ rm newname.txt
j:~/Week.8/8.1.The.Command.Line$ ls
Supplement.8.1.pdf   test_dir  Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$
```

After running the `rm` command, `newname.txt` is no longer in the directory. To remove the entire `test_dir` directory the `-r` option is required. First, try removing the directory without the `-r` option:

```
j:~/Week.8/8.1.The.Command.Line$ ls
Supplement.8.1.pdf   test_dir  Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$ rm test_dir
rm: cannot remove 'test_dir': Is a directory
j:~/Week.8/8.1.The.Command.Line$ ls
Supplement.8.1.pdf   test_dir  Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$
```

An error message was output, and the directory was not removed. Now remove the `test_dir` directory with the `-r` option:

```
j:~/Week.8/8.1.The.Command.Line$ ls
Supplement.8.1.pdf   test_dir  Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$ rm -r test_dir
j:~/Week.8/8.1.The.Command.Line$ ls
Supplement.8.1.pdf   Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$
```

Notice that both test_dir **and** all its contents have been deleted.

Warning: There is no undo in the command line. The `rm` command removes files and directories permanently, there is no recycling or trash bin from which files can be recovered. Always double check commands before pressing enter.

Files can also be lost when using the `cp` and `mv` commands. If a file exists with the same path and name as the DEST argument, the file will be overwritten. For example, say there is a directory containing two files, `patients.txt` and `doctors.txt`, which contain patient information and doctor information, respectively. If the following command is run:

```
cp patients.txt doctors.txt
```

The file `patients.txt` will still contain the patient information, but the file `doctors.txt` will now also contain the patient information but will still have the name

`doctors.txt`. The information contained in doctors.txt has been **permanently** overwritten and lost.

Similarly, say there is another directory containing two files, `genes.txt` and `proteins.txt`, which contain a list of genes and a list of proteins, respectively. If the following command is run:

```
mv genes.txt proteins.txt
```

There will now only be one file, which will be called `proteins.txt`, however it will contain the contents of `genes.txt`.

Always double check your commands and create back-up copies of important files in case something goes wrong. For example, in the home directory you can create a new directory called `file_backup` and copy any important files into it.

```
j:~/Week.8/8.1.The.Command.Line$ mkdir /home/jovyan/file_backup
j:~/Week.8/8.1.The.Command.Line$ cp important_file.txt ../../file_backup
j:~/Week.8/8.1.The.Command.Line$
```

The Wildcard

A handy feature of the command line is the **wildcard**. The wildcard, represented by an asterisk (*) can represent any characters, any number of times.

Copy the following commands and run them in the Terminal to create a directory that can be used to examine the uses of the wildcard. The touch command creates empty files

```
j:~/Week.8/8.1.The.Command.Line$ mkdir test_wildcard
j:~/Week.8/8.1.The.Command.Line$ cd test_wildcard
j:~/Week.8/8.1.The.Command.Line/test_wildcard$ touch file_A.txt
file_B.txt info_A.csv info_B.csv
j:~/Week.8/8.1.The.Command.Line/test_wildcard$
```

You should now be in the `8.1.The.Command.Line/test_wildcard` directory, which contains 4 files.

```
j:~/Week.8/8.1.The.Command.Line/test_wildcard$ ls
file_A.txt  file_B.txt  info_A.csv  info_B.csv
j:~/Week.8/8.1.The.Command.Line/test_wildcard$
```

As stated above, the wildcard can act as a representation of multiple characters. Thus, `*.txt` can be used to represent any file that ends in `.txt`. To list all the files in the `test_wildcard` directory, `*.txt` can be used with the `ls` command.

```
j:~/Week.8/8.1.The.Command.Line/test_wildcard$ ls *.txt
file_A.txt  file_B.txt
j:~/Week.8/8.1.The.Command.Line/test_wildcard$
```

The wildcard can also be used to list all the files that contain the letter A somewhere in the file name by putting a wildcard before and after the letter A. This means that any number of characters can occur before the A, and any number can occur after.

```
j:~/Week.8/8.1.The.Command.Line/test_wildcard$ ls *A*
file_A.txt  info_A.csv
j:~/Week.8/8.1.The.Command.Line/test_wildcard$
```

The wildcard can be used with other commands as well. To copy all the files that start with the word "info" into the parent directory (`8.1.The.Command.Line`), the following `cp` command can be executed:

```
j:~/Week.8/8.1.The.Command.Line/test_wildcard$ ls ..
Supplement.8.1.pdf   test_wildcard  Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line$ cp info* ..
j:~/Week.8/8.1.The.Command.Line$ ls ..
info_A.csv  info_B.csv  Supplement.8.1.pdf   test_wildcard  Tutorial.8.1
j:~/Week.8/8.1.The.Command.Line/test_wildcard$
```

When the contents of the parent directory (`8.1.The.Command.Line`) are listed after the `cp` command is executed the two info files, `info_A.csv` and `info_B.csv`, have been copied over.