Command Line

`Data Science`

- Command Line
  - Change directory
    - Option
    - Copying
      - Copy 1 file
      - Copy multiple files
      - Wildcards
    - Moving
      - Moving 1 and multiple
      - Renaming Files
    - Removing Files/Directories
  - Redirection (I/O)
      - > Command
      - >> Command
      - < Commandr
      - | Pipe
      - Sort
      - Uniq
      - grep
      - sed
    - Redicrction Summary
  - Environment
    - nano
      - Creating a file
      - Creating a File to Store Environment Setting
      - Environment env
    - Summary

Change directory

```
cd .. # go back
cd ../feb # go back and then to /feb
mkdir media # create new directory
touch keyboard.txt # create new file
```

- The *command line* is a text interface for the computer's operating system. To access the command line, we use the terminal.
- A *filesystem* organizes a computer's files and directories into a tree structure. It starts with the *root directory*. Each parent directory can contain more child directories and files.
- From the command line, you can navigate through files and folders on your computer:
  - `pwd` outputs the name of the current working directory.
  - `ls` lists all files and directories in the working directory.
  - `cd` switches you into the directory you specify.
  - `mkdir` creates a new directory in the working directory.
  - `touch` creates a new file inside the working directory.

Option

```
$ls
$ls -a # list including all hidden files
$ls -l # lists all contents of a directory in long format
$ls -t # order files and directories by the time they were last mod
ified
$ls -alt # combine the function of -a, -l, -t together.
```

`$ls -l`

1. Access rights. These are actions that are permitted on a file or directory.

2. Number of hard links. This number counts the number of child directories and files. This number includes the parent directory link (..) and current directory link (.).
3. The username of the file's owner. Here the username is cc.
4. The name of the group that owns the file. Here the group name is eng.
5. The size of the file in bytes.
6. The date & time that the file was last modified.
7. The name of the file or directory.

Copying

- Options modify the behavior of commands:
  - `ls -a` lists all contents of a directory, including hidden files and directories
  - `ls -l` lists all contents in long format
  - `ls -t` orders files and directories by the time they were last modified
  - Multiple options can be used together, like `ls -alt`
- From the command line, you can also copy, move, and remove files and directories:
  - `cp` copies files
  - `mv` moves and renames files
  - `rm` removes files
  - `rm -r` removes directories
- Wildcards are useful for selecting groups of files and directories

Copy 1 file

```
cp biopic/cleopatra.txt historical/
```

To copy a file into a directory, use `cp` with the source file as the first argument and the destination directory as the second argument. Here, we copy the file **biopic/cleopatra.txt** and place it in the **historical/ directory**.

Copy multiple files

```
cp biopic/ray.txt biopic/notorious.txt historical/
```

Wildcards

`*` selects all the files in the working directories.

```
cp * satire/
cp m*.txt scifi/ #selects all files in the working directory starti
ng with "m" and ending with ".txt", and copies them to scifi/.
```

Moving

Moving 1 and multiple

```
mv superman.txt superhero/ # move to file superhero
mv wonderwoman.txt batman.txt superhero/ # move multiple to file su
perhero
```

Renaming Files

```
mv batman.txt spiderman.txt # rename batman.txt as spiderman.txt
```

Removing Files/Directories

```
rm waterboy.txt # remove from the current directory
rm -r slapstick # remove a directory
```

Redirection (I/O)

1. *standard input*, abbreviated as `stdin`, is information inputted into the terminal through the keyboard or input device.
2. *standard output*, abbreviated as `stdout`, is the information outputted after a process is run.
3. *standard error*, abbreviated as `stderr`, is an error message outputted by a failed process.

Redirection reroutes standard input, standard output, and standard error to or from a different location.

take string as *standard input* and give *standard output*

```
echo "Hello"
```

`>` Command

```
$ echo "Hello" > hello.txt
$ cat hello.txt
```

The *standard output* `"Hello"` is redirected by `>` to the file **hello.txt**.

```
cat oceans.txt > continents.txt
```

Here the standard output of `cat oceans.txt` is redirected to `continents.txt`. `>` will overwrite existing material. (区别于`>>`)

`>>` Command

```
$ cat glaciers.txt >> rivers.txt
```

`>>` takes the standard output of the command on the left and ***appends (adds)*** it to the file on the right.

`<` Commandr

```
$ cat < lakes.txt
```

`<` takes the standard input from the file on the right and inputs it into the program on the left. Here, lakes.txt is the standard input for the cat command. The standard output appears in the terminal.

`|` Pipe

```
$ cat volcanoes.txt | wc # wc command outputs the number of lines, words, and characters in volcanoes.txt, respectively.
```

Multiple Pipes

```
cat volcanoes.txt | wc | cat > islands.txt
```

Sort

```
$ cat lakes.txt
$ cat lakes.txt | sort > sorted-lakes.txt
```

Uniq

```
$ uniq deserts.txt
$ sort deserts.txt | uniq
$ sort deserts.txt | uniq > uniq-deserts.txt
```

`grep`

“global regular expression print”

```
$ grep Mount mountains.txt # every line that contains "Mount"
$ grep -i Mount mountains.txt # every line that contains "Mount" - case insensitive
```

```
$ grep -R Arctic /home/ccuser/workspace/geography
```

`grep -R` searches all files in a directory and outputs filenames and lines containing matched results. `-R` stands for “recursive”. Here grep `-R` searches the /home/ccuser/workspace/geography directory for the string “Arctic” and outputs filenames and lines with matched results.

```
$ grep -Rl Arctic /home/ccuser/workspace/geography
```

outputs only filenames with matched results.

`sed`

`sed` stands for “stream editor”. It accepts standard input and modifies it based on an *expression*, before displaying it as output data. It is similar to “find and replace”.

```
$ cat forests.txt
$ sed 's/snow/rain/' forests.txt
$ sed 's/snow/rain/g' forests.txt
```

`s`: stands for “substitution”. it is always used when using `sed` for substitution.
`snow`: the search string, the text to find.
`rain`: the replacement string, the text to add in place.
`g`: globally, w/o it, only the first ‘snow’ will be replaced by ‘rain’ **in a line**.

```
$ cat forests.txt
The Amazon snowforest
The Congo snowforest
Valdivian Temperate snowforest
Daintree snowforest
Southeast Asian snowforest snowforest
Tongrass National forest
Sinharaja Forest Reserve
Pacific Temperate snowforest snowforest

$ sed 's/snow/rain/' forests.txt
The Amazon rainforest
The Congo rainforest
Valdivian Temperate rainforest
Daintree rainforest
Southeast Asian rainforest snowforest
Tongrass National forest
Sinharaja Forest Reserve
Pacific Temperate rainforest snowforest

$ sed 's/snow/rain/g' forests.txt
The Amazon rainforest
The Congo rainforest
Valdivian Temperate rainforest
Daintree rainforest
Southeast Asian rainforest rainforest
Tongrass National forest
Sinharaja Forest Reserve
Pacific Temperate rainforest rainforest
```

Redicrction Summary

- *Redirection* reroutes standard input, standard output, and standard error.
- The common redirection commands are:
  - `>` redirects standard output of a command to a file, overwriting previous content.
  - `>>` redirects standard output of a command to a file, appending new content to old content.
  - `<` redirects standard input to a command.
  - `|` redirects standard output of a command to another command.
- A number of other commands are powerful when combined with redirection commands:
  - `sort` : sorts lines of text alphabetically.
  - `uniq` : filters duplicate, adjacent lines of text.
  - `grep` : searches for a text pattern and outputs it.
  - `sed` : searches for a text pattern, modifies it, and outputs it.

Environment

We can configure the environment to support the commands and programs we create. This enables us to customize greetings and command aliases, and create variables to share across commands and programs.

`nano`

Creating a file

```
$ nano hello.txt
```

In nano, at the top of the window, type

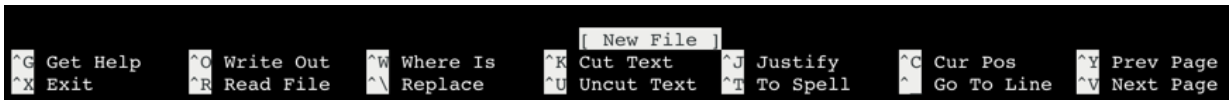
```
"Hello, I am nano."
```

Using the menu at the bottom of the terminal for reference, type `Ctrl` + `O` (the letter, not the number) to save the file. This is the letter "O", not the number zero.

Press `Enter`, when prompted about the filename to write.

Then type `Ctrl` + `X` to exit nano.

Finally, type `clear` to clear the terminal window. The command prompt should now be at the top of the window.




```
$ cat hello.txt # print "Hello, I am nano"
```

Creating a File to Store Environment Setting

```
$ nano ~/.bash_profile # create a file called ~/.bash_profile
```

In **~/.bash_profile**, at the top of the file, type

```
echo "Welcome, Jane Doe"
```

You can use your name in place of "Jane Doe".

Type `Ctrl` + `O` to save the file.

Press `Enter` to write the filename.

Type `Ctrl` + `X` to exit.

Finally, type `clear` to clear the terminal window.

```
$ source ~/.bash_profile # print "Hello, Jane Doe"
```

```
$ nano ~/.bash_profile
```

~/.bash_profile is the name of file used to store environment settings. It is commonly called the "bash profile". When a session starts, it will load the contents of the bash profile before executing commands.

- The `~` represents the user's home directory.
- The `.` indicates a hidden file.
- The name **~/.bash_profile** is important, since this is how the command line recognizes the bash profile.

1. The command `nano ~/.bash_profile` opens up **~/.bash_profile** in nano.
2. The text `echo "Welcome, Jane Doe"` creates a greeting in the bash profile, which is saved. It tells the command line to `echo` the string "Welcome, Jane Doe" when a terminal session begins.
3. The command `source ~/.bash_profile` activates the changes in **~/.bash_profile** for the current session. Instead of closing the terminal and needing to start a new session, `source` makes the changes available right away in the session we are in.

```
nano ~/.bash_profile # open ~/.bash_profile so we can keep editing
```

In **~/.bash_profile**, beneath the greeting you created, type

```
alias pd="pwd"
```

```
$ pd # print directory
```

The `alias` command allows you to create keyboard shortcuts, or aliases, for commonly used commands.

1. Here `alias pd="pwd"` creates the alias `pd` for the `pwd` command, which is then saved in the bash profile. Each time you enter `pd`, the output will be the same as the `pwd` command.
2. The command `source ~/.bash_profile` makes the alias `pd` available in the current session.

In the bash profile, beneath the previous alias, add

```
alias hy="history"
```

Add another alias

```
alias ll="ls -la"
```

```
$ hy # print history
$ ll # print "$ ls -la" of the current directory
```

```
alias hy="history"
```

`hy` is set as alias for the `history` command in the bash profile. The alias is then made available in the current session through `source`. By typing `hy`, the command line outputs a history of commands that were entered in the current session.

```
alias ll="ls -la"
```

`ll` is set as an alias for `ls -la` and made available in the current session through `source`. By typing `ll`, the command line now outputs all contents and directories in long format, including all hidden files.

In the bash profile, beneath the aliases, on a new line, type

```
export USER="Jane Doe"
```

Therefore, the current `~/.bash_profile` we have is:

```
echo "Welcome, Howell Yu"
alias pd="pwd"
alias hy="history"
alias ll="ls -la"
export USER="Howell Yu"
```

```
$ echo $USER
```

- Sets the environment variable USER to a name “Howell Yu”
- Export makes the variable to be available to all child sessions initiated from the session you are in.

On a new line, beneath the last entry, type

```
export PS1=">> "
```

`PS1` is a variable that defines the makeup and style of the command prompt.

1. `export PS1=">> "` sets the command prompt variable and exports the variable. Here we change the default command prompt from `$` to `>>`.
2. After using the `source` command, the command line displays the new command prompt.

```
>> echo $HOME # print "/home/ccuser"
```

```
echo $PATH
```

Environment `env`

```
$ env
```

Returns a list of variables for the current users.
i.e. including `PATH`, `PWD`, `PS1`, and `HOME`.

```
$ env | grep PATH
```

Here the standard output of `env` is "piped" to the `grep` command. `grep` searches for the value of the variable `PATH` and outputs it to the terminal.
i.e. It displays the value of a single environment variable

Summary

- The *environment* refers to the preferences and settings of the current user.
- The *nano* editor is a command line text editor used to configure the environment.
- **~/.bash_profile** is where environment settings are stored. You can edit this file with nano.
- *environment variables* are variables that can be used across commands and programs and hold information about the environment.
    - `export VARIABLE="Value"` sets and exports an environment variable.
    - `USER` is the name of the current user.
    - `PS1` is the command prompt.
    - `HOME` is the home directory. It is usually not customized.
    - `PATH` returns a colon separated list of file paths. It is customized in advanced cases.
    - `env` returns a list of environment variables.