Module 2 Pre-Read - command line

Saturday, February 3, 2024 11:45

Shift+Up/Down Arrow: Scrolling

Ctrl+L: Move cursor to top of the screen

Highlighting text: Copies the text to clipboard

In a shell script in an IDE: ctrl+enter will send it to the terminal

Navigation using the terminal

Commands:

`pwd`: print working directory

`ls`: see all the objects in that working directory

`cd:` How to move to a subfolder within a folder

absolute file path: `/c/Users/Cathy`

Making directories and files in terminal

`mkdir foldername`: create a directory/folder called foldername

`touch code/test_code3.R`: adds an empty .R file called test_code3 in the subdirectory folder called code

`rm test_code3.R`: removes ALL TRACES of the file from the directory

Copying and moving files in terminal

`cp myfile.md code/`: copy a file called myfile.md into the subdirectory code/

`cp myfile.md newname.md`: copy the file but under a different name

`rm newname.md`: removes ALL TRACES of the file from the directory

`rm code/myfile.md`: removes ALL TRACES of the file from the subdirectory called code

`mv myfile.md code/` : move myfile.md into the subdirectory called code

`mv myfile.md ..` : move the file UP a directory (out of the subdirectory)

Terminal shortcuts

`cd ..` : move up a directory (out of the subdirectory) `cd ../..` : move up TWO directories

`cd .` : change directory into current directory (nothing should happen)

`echo $HOME` : shows what my home directory is. Each user has a different home directory.

`cd ~` : puts me back into the home directory

<u>Absolute vs. relative filepaths</u>

Absolute filepath: reference one specific location on YOUR computer, starts from the root directory e.g. `/c/Users/Cathy/Pictures/Wallpapers`

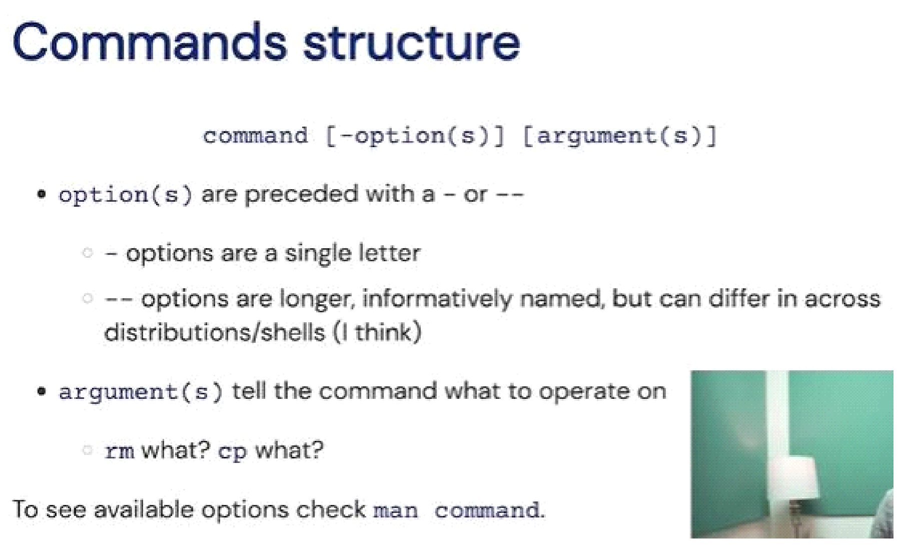
Shortcut: `~/Pictures/Wallpapers`

Relative filepath: depends on what current working directory you are in e.g. `cd code` depends on you having a code folder in the current directory you're in

We want to rely on relative filepaths for reproducible code because we want the code to run on multiple people's computers

<u>Structure of bash commands</u>

`rm -rf myfolder` : (risky move) gets rid of the entire directory or folder called myfolder. the `-r` means "recursive" and `-f` means "force"



Commands structure

```
command [-option(s)] [argument(s)]
```

- `option(s)` are preceded with a - or --
  - - options are a single letter
  - -- options are longer, informatively named, but can differ in across distributions/shells (I think)
- `argument(s)` tell the command what to operate on
  - `rm` what? `cp` what?

To see available options check `man command`.

in git bash, we type `command --help` instead of `man command`

<u>Wildcards</u>

`ls test_code?.R` : lists all of the files with the name test_code and any character where the ? is, and then finish with a .R, and each ? corresponds to a single character `ls test_code[1-3].R` : looks for the range of files test_code1.R test_code2.R test_code3.R

`ls *.R` : list all the .R files.

`ls test_code1*` : list all the files that start with test_code1

`ls *_*.R` : list all the files that include an _ and then is also a .R file

Defining variables in bash

`my_variable="hello world"` : Creates a variable called my_variable with a string called "hello world". NO SPACES, otherwise bash thinks it's a command not a variable

`echo $my_variable` : returns the value of the variable

`echo "A nice greeting in computing is $my_variable"` : will print A nice greeting in computing is hello world

`echo "A nice greeting in computing is {my_variable}asfasdfasdf"` : will print A nice greeting in computing is helloworldasfasdfasdf

Command substitution

`my_sequence=$(seq 1 10)` : run a command and save the output a command as a variable (here it is a sequence of 1 2 3 4 5 6 7 8 9 10)

For loops and if/else statements in bash

```
for name in Winn Nick Weiwei

do

  echo "My name is $name"

  echo "No, really, my name is $name"

done
```

This will print out:

```
    My name is Winn

    No, really, my name is Winn

    My name is Nick

    No, really, my name is Nick

    My name is Weiwei

    No, really, my name is Weiwei
```

```
for i in $(seq 1 10)
```

```
do
echo "This is iteration $i"
done
```

This will print out

```
    This is iteration 1
    This is iteration 2
    This is iteration 3
    This is iteration 4
    This is iteration 5
    This is iteration 6
    This is iteration 7
    This is iteration 8
    This is iteration 9
    This is iteration 10
```

```
for i in $(seq 1 10)
do
  if [ "$i" == "3" ]
  then
    echo "This is the best iteration ever"
  else
    echo "This is iteration $i"
  fi
done
```

This will print out

```
    This is iteration 1
    This is iteration 2
    This is the best iteration ever
    This is iteration 4
    This is iteration 5
    This is iteration 6
    This is iteration 7
    This is iteration 8
    This is iteration 9
    This is iteration 10
```

```
for i in $(seq 1 10)
do
  if [ "$i" == "3" ]
  then
    echo "This is the best iteration ever"
  elif [ "$i" == "5" ]
  then
    echo "This is the worst iteration ever"
  else
    echo "This is iteration $i"
  fi
done
```

This will print out

```
    This is iteration 1
    This is iteration 2
    This is the best iteration ever
    This is iteration 4
    This is the worst iteration ever
    This is iteration 6
    This is iteration 7
    This is iteration 8
    This is iteration 9
    This is iteration 10
```

<u>Viewing file contents from the command line</u>

`cat myfile.R` : will print what is in the file myfile. Will clutter up your console for larger files

`less myfile.R` : we'll move into another interactive browser (press `q` to exit)

Redirecting output from files and searching for text

`curl -L http://bit.ly/hamlet_txt`: will go to the website and print out all the text from that website

`curl -L http://bit.ly/hamlet_txt> hamlet.txt` : will go to the website and put all the text into a file called hamlet.txt

`echo "Hello world" > hello_world.txt` : will redirect "Hello world" into a new file called hello_world.txt, will also overwrite anything in hello_world.txt if it exists already

`echo "Hello world again" >> hello_world.txt` : will add "Hello world again" into the file called hello_world.txt

`grep "Ham\." hamlet.txt` : will return every line in hamlet.txt with "Ham." (need the backslash for the period because the period is a special character)

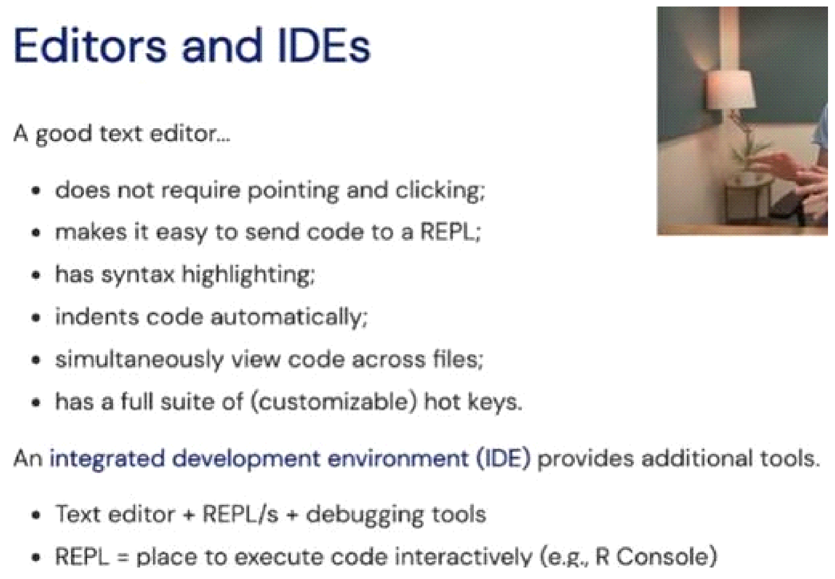
Piping in bash

`grep "Ham\." hamlet.txt | wc -l` : take the output of `grep "Ham\." hamlet.txt` and count the lines (`wc` for wordcount, `-l` for line)

`grep "Ham\." hamlet.txt | head -5` : take the output of `grep "Ham\." hamlet.txt` and shows the first five lines (use `tail -1` for last line)

Using text editors and IDEs

IDE: Integrated Development Environment



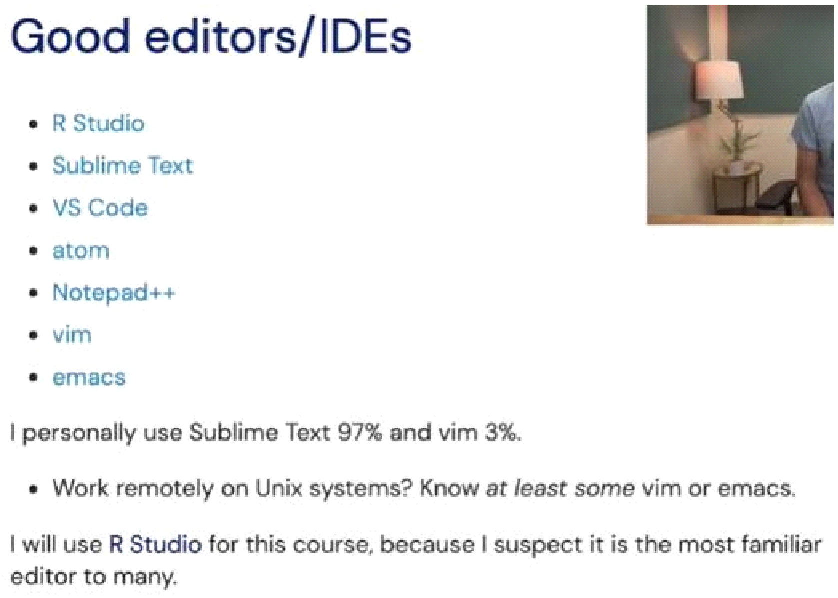
Editors and IDEs

A good text editor...

- does not require pointing and clicking;
- makes it easy to send code to a REPL;
- has syntax highlighting;
- indents code automatically;
- simultaneously view code across files;
- has a full suite of (customizable) hot keys.

An integrated development environment (IDE) provides additional tools.

- Text editor + REPL/s + debugging tools
- REPL = place to execute code interactively (e.g., R Console)



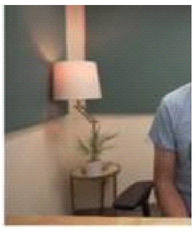
Good editors/IDEs

- R Studio
- Sublime Text
- VS Code
- atom
- Notepad++
- vim
- emacs

I personally use Sublime Text 97% and vim 3%.

- Work remotely on Unix systems? Know *at least some* vim or emacs.

I will use R Studio for this course, because I suspect it is the most familiar editor to many.



Shell scripts

In R, click into the terminal tab. You can write shell script like normal in the code part in your script, and then send it to the terminal.

To run the code in the script, type:

```
bash filename.sh
```

this will run everything that is in that file

Shebangs (#!)

`#! /usr/bin/bash` in the script says that everything after it should be run using bash, so you don't need to say `bash filename.sh`, just say `./filename.sh`

`which bash` for the location of that bash thingy

A little less reproducible.