Introduction to shell for Data science

pwd = print working directory

/home/repl/course.txt : absolute path

ls = list of objects at a place (ls /home/repl/seasonal gives all in the folder seasonal)

relative path: location seen from where you are right now. seasonal/winter.csv vs /home/repl/seasonal/winter.csv)

cd = change directory

cd .. = will move you one

. (single dot) current directory

~ = your home directory

cp = copy a file (e.g. cp seasonal/autumn.csv seasonal/winter.csv backup -> will copy the two csv files to the folder backup

mv = move a file

rm = remove a file

rmdir = can delete an entire folder / directory if it is impty 🡪 delete files first

mkdir = create a new directory

cat = print the content of a file

less = print a file pagewise press spacebar to go down a site or :q to quit

you can less multiple files… us :n to switch to the next file

csv = comma spreaded values

head = look at the first 10 lines of a vile

use tab to write out the path automatically

You won't always want to look at the first 10 lines of a file, so the shell lets you change head's behavior by giving it a **command-line flag** (or just "flag" for short). If you run the command:

head -n 3 seasonal/summer.csv

head will only display the first three lines of the file. If you run head -n 100, it will display the first 100 (assuming there are that many), and so on.

Ls –R = prints everything underneath the directory

Ls –F = prints name and every runnable program underneath directory

man = manual or help for a function (invokes less 🡪 use spacebar and :q to quit

cut = select rows from a file It has several options (use man cut to explore them), but the most common is something like:

cut -f 2-5,8 -d , values.csv

which means "select columns 2 through 5 and columns 8, using comma as the separator". cut uses -f (meaning "fields") to specify columns and -d (meaning "delimiter") to specify the separator. You need to specify the latter because some files may use spaces, tabs, or colons to separate columns.

history = prints a list of recently used commands

!Number =  type !55 to re-run the 55th command in your history. You can also re-run a command by typing an exclamation mark followed by the command's name, such as !head or !cut, which will re-run the most recent use of that command.

grep = selects lines accorging to what they contain For example, grep bicuspid seasonal/winter.csv prints lines from winter.csvthat contain "bicuspid".

grep can search for patterns as well; we will explore those in the next course. What's more important right now is some of grep's more common flags:

-c: print a count of matching lines rather than the lines themselves

-h: do *not* print the names of files when searching multiple files

-i: ignore case (e.g., treat "Regression" and "regression" as matches)

-l: print the names of files that contain matches, not the matches

-n: print line numbers for matching lines

-v: invert the match, i.e., only show lines that *don't* match

Paste = merge lines of files

> name = save a file : head -n 5 seasonal/summer.csv > top.csv

| = the pipe command takes the output from the previous command and redirects it to the next eg: head -n 5 seasonal/summer.csv | tail -n 3

Wc = word count prints the number of characters, words, and lines in a file. You can make it print only one of these using -c, -w, or -l respectively.

Choose multiple files:

cut -d , -f 1 seasonal/winter.csv seasonal/spring.csv seasonal/summer.csv seasonal/autumn.csv

or

cut -d , -f 1 seasonal/\*

or cut -d , -f 1 seasonal/\*.csv

The most common wildcard is \*, which means "match zero or more characters". Other wildcards:

* ? matches a single character, so 201?.txt will match 2017.txt or 2018.txt, but not 2017-01.txt.
* [...] matches any one of the characters inside the square brackets, so 201[78].txt matches 2017.txt or 2018.txt, but not 2016.txt.
* {...} matches any of the comma-separated patterns inside the curly brackets, so {\*.txt, \*.csv} matches any file whose name ends with .txt or .csv, but not files whose names end with .pdf.

Sort = sort data (ascending alphabetic) -n and -r can be used to sort numerically and reverse the order of its output, while -b tells it to ignore leading blanks and -f tells it to **f**old case (i.e., be case-insensitive)

uniq = remove adjacient douplicate lines

Ctrl + C or ^C can stop a running calculation

Like other programs, the shell stores information in variables. Some of these, called **environment variables**, are available all the time. Environment variables' names are conventionally written in upper case, and a few of the more commonly-used ones are shown below.

HOME; PWD; SHELL; USER;

echo = prints the value of its arguments

echo X prints the name of x, echo $x prints the value of x

The other kind of variable is called a **shell variable**, which is like a local variable in a programming language. To create a shell variable, you simply assign a value to a name:

training=seasonal/summer.csv

without any spaces before or after the = sign. Once you have done this, you can check the variable's value

for filetype in gif jpg png; do echo $filetype; done

it produces:

gif

jpg

png

Notice these things about the loop:

1. The structure is for ...variable... in ...list... ; do ...body... ; done
2. The list of things the loop is to process (in our case, the words gif, jpg, and png).
3. The variable that keeps track of which thing the loop is currently processing (in our case, filetype).
4. The body of the loop that does the processing (in our case, echo $filetype).

Examples:

for file in seasonal/\*.csv; do head -n 2 $file | tail -n 1; done

for f in seasonal/\*.csv; do echo $f; head -n 2 $f | tail -n 1; done

Unix has a bewildering variety of text editors. For this course, we will use a simple one called Nano. If you type nano filename, it will open filename for editing (or create it if it doesn't already exist). You can move around with the arrow keys, delete characters using backspace, and do other operations with control-key combinations:

* Ctrl + K: delete a line.
* Ctrl + U: un-delete a line.
* Ctrl + O: save the file ('O' stands for 'output').
* Ctrl + X: exit the editor.
* Note that in Nano, "copy and paste" is achieved by navigating to the line you want to copy, pressing CTRL + K to cut the line, then CTRL + U twice to paste two copies of it.

Commands can be saved as files to be rerun by calling the commandfile using bash:

Use nano dates.sh to create a textfile with a command like

cut -d , -f 1 seasonal/\*.csv

then use bash dates.sh to run the command

Shell scripts can also contain loops. You can write them using semi-colons, or split them across lines without semi-colons to make them more readable:

# Print the first and last data records of each file.

for filename in $@

do

head -n 2 $filename | tail -n 1

tail -n 1 $filename

done

(You don't have to indent the commands inside the loop, but doing so makes things clearer.)

The first line of this script is a **comment** to tell readers what the script does. Comments start with the # character and run to the end of the line. Your future self will thank you for adding brief explanations like the one shown here to every script you write.