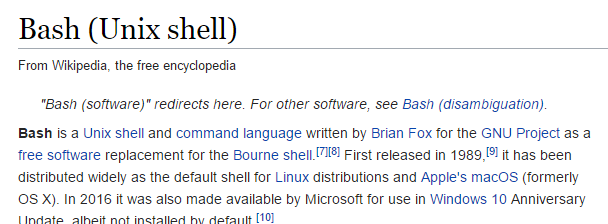
BASH SHELL

BASH, an acronym for Bourne-Again Shell, is a command language that allows a data scientist to manage files “outside” of a software package. BASH allows you to manage files “at the operating system level”. You can also create and execute shell scripts using BASH – shell scripts can be used to change or influence the configuration of systems installed on your computer.



There are different version of BASH that exist. The version that we will be using in GNU BASH. The following provides instructions on how to install GNU Bash on PC/MAC machines.

|  |  |
| --- | --- |
| Image result for gnu bash | |
| Computer | Process / Links |
| PC Options | * Use Git Bash which comes pre-packaged with a Git installation. Download Git from <https://git-scm.com/>   Git Bash  Image result for google images git bash  After installing git, type Git Bash in your program search box. The Git Bash desktop app will open.   * The Windows 10 Enterprise operating system does contain BASH on Ubuntu. To install see: <http://www.windowscentral.com/how-install-bash-shell-command-line-windows-10>   BASH on Ubuntu  Image result for icon windows 10 bash |
| MAC Option | * Step 1: Install Homebrew.   Select Applications > Utilities > Terminal  Image result for mac os terminal icon  Type the following onto a single line in the terminal window. You may choose to visit <http://brew.sh> and this line can be copied and pasted into the terminal window. You will need to provide a password during the installation process.  /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"   * Step 2: Install gnu-sed using Homebrew. Type the following at the command prompt.   brew install gnu-sed |
|  |  |

The BASH window has the following appearance.



Before using BASH to manage files, let us first understand how to move around inside of BASH.

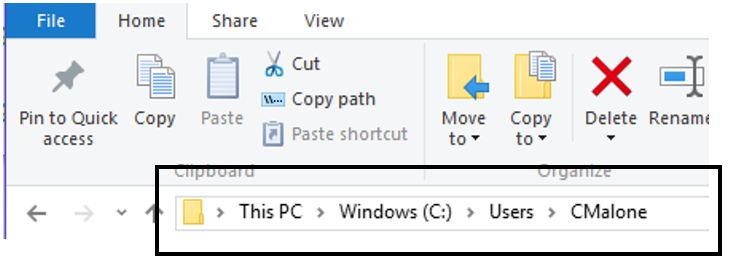
Consider the command that identifies your current location.

|  |  |
| --- | --- |
| Command | Action |
| $pwd | Get path of working directory |

$ pwd

/c/Users/CMalone

The path provided here is the same path used in Windows Explorer.



|  |
| --- |
| **Setting up a folder with contents**  Create a folder named DSCI210 inside the directory specified above. Create a subdirectory called Data. From the course website, download the Babies\_2014.txt and Babies\_2013.txt data files and place them inside the Data subdirectory. |

MOVING AROUND

The change directory command can be used to move down (or up) the directory tree-structure.

|  |  |
| --- | --- |
| Command | Action |
| cd <folder name> | Move into a folder |
| cd .. | Move up a folder in the directory tree |

Move into the DSCI210 folder with the following command.

$ cd DSCI210

Next, moving into the Data folder.

$ cd Data

Realize, this could the above two commands can be done in a single step.

$ cd DSCI210/Data

Identify your current location.

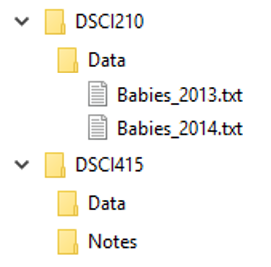
$ pwd

/c/Users/CMalone/DSCI210/Data

Suppose my current location is within the Data folder and you want to move into the DSCI415/Notes folder.

|  |  |
| --- | --- |
| Current location | /c/Users/CMalone/DSCI210/Data |
| Desired location | /c/Users/CMalone/DSCI415/Notes |

$ cd ../../DSCI415/Notes



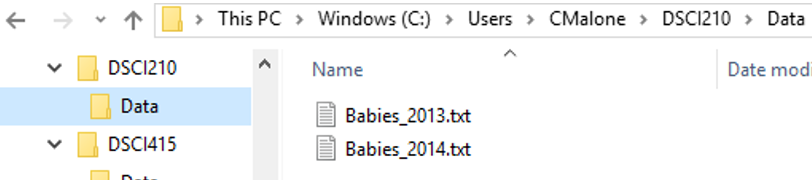
|  |  |  |  |
| --- | --- | --- | --- |
| Command | Move out of DSCI210/Data | Move out of DSCI210 | Move into DSCI415/Notes |
| $ cd ../../DSCI415/Notes | ../ | ../ | DSCI415/Notes |

The movement into another location can be done using the complete path as well.

$ cd /c/Users/CMalone/DSCI415/Notes

LOOKING IN DIRECTORIES / FILES

The usual method of looking in directories is to click your way through the directory tree-structure. Likewise, the look at a file, we normally double-click and the file will open in the particular software package that is associated with that file type.



The ls command can be used to identify the contents of the current directory.

|  |  |
| --- | --- |
| Command | Action |
| ls | List contents of folder |
| ls – l | With the – l option, additional file information is provided |
| cat | Print contents of file to screen |

$ ls

Babies\_2013.txt Babies\_2014.txt

The ls command with the –l option provides additional details.

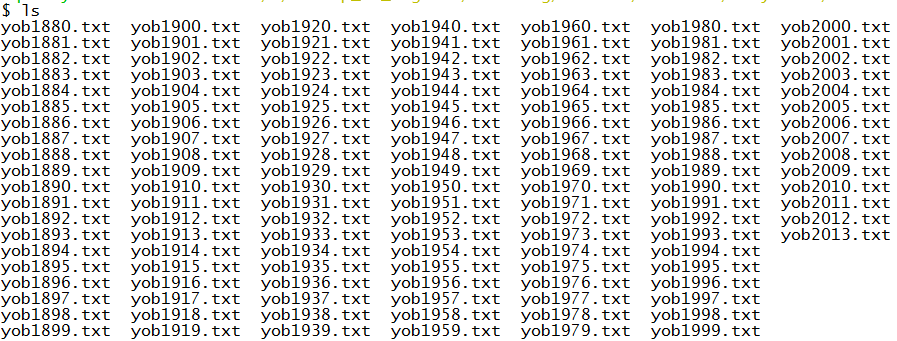
$ ls -l

total 760

-rw-r--r-- 1 aq7839yd 1049089 370653 Feb 22 09:24 Babies\_2013.txt

-rw-r--r-- 1 aq7839yd 1049089 404752 Feb 22 09:24 Babies\_2014.txt

Next, consider a situation in which management of hundreds of files is required. For simplicity, suppose all files sit in same directory. In this situation, it may be nice to get a list of all the files contained in this directory. That is, instead of pushing the output from the ls command to the screen, it can be pushed into a file.



|  |  |
| --- | --- |
| Command | Action |
| … > <filename.txt> | Put contents from command into file instead of printing to screen |

The following can be used to save the contents from the ls command into a file called Contents.txt

$ ls > Contents.txt

The ls command to make sure the Content.txt was created successfully.

$ ls

Babies\_2013.txt Babies\_2014.txt Content.txt

Looking at the content of this newly created file using the cat command.

$ cat Content.txt

Babies\_2013.txt

Babies\_2014.txt

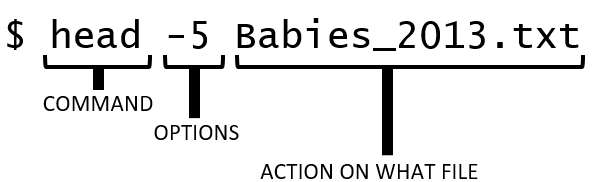
Content.txt

|  |  |
| --- | --- |
| **Aside:** The newline characters varies between operating systems. In BASH, it appears that the Contents.txt file has three lines; however, when the file is viewed in Notepad (the standard text editor in Windows), this file appears to only have a single line. Worse, there is no delimiter between the file names. | |
|  | |
| The line feed, i.e. \n, character as shown in Notepad++. | The line feed character is \*not\* sufficient in Windows for a new line. Windows also needs the hidden carriage return. |
|  |  |
| Some software programs, e.g. Sublime, are written to automatically identify either type of newline characters. However, some software, e.g. Notepad, is written to only accept their respective version of the newline character. | |

The cat command prints all lines from the file to the screen. The head/tail commands can be used to see only the top/bottom lines in a file.

|  |  |
| --- | --- |
| Command | Action |
| head –n | Show top n lines of file |
| tail -n | Show bottom n lines of file |

BASH commands have the following general structure.



Getting the first few lines of Babies\_2013.txt using the head command.

$ head Babies\_2013.txt

"Notes" "County" "County Code" "Month" "Month Code" Births

"Baldwin County, AL" "01003" "January" "1" 200

"Baldwin County, AL" "01003" "February" "2" 165

"Baldwin County, AL" "01003" "March" "3" 163

"Baldwin County, AL" "01003" "April" "4" 190

"Baldwin County, AL" "01003" "May" "5" 157

"Baldwin County, AL" "01003" "June" "6" 173

"Baldwin County, AL" "01003" "July" "7" 199

"Baldwin County, AL" "01003" "August" "8" 198

"Baldwin County, AL" "01003" "September" "9" 157

Getting the last 3 lines of the Babies\_2013.txt file.

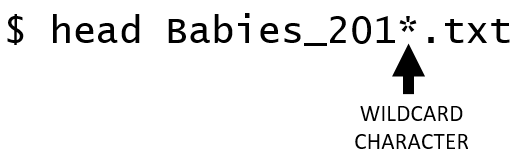
$ tail -3 Babies\_2013.txt

"10. New York County, New York (FIPS code 36061) represents Manhattan Borough, New York City."

"11. Queens, New York (FIPS code 36081) represents Queens Borough, New York City."

"12. Richmond County, New York (FIPS code 36085) represents Staten Island Borough, New York City."

BASH has the ability to work with wildcard characters. Suppose you wanted to see the first few lines of both Babies\_2013.txt and Babies\_2014.txt. The wildcard character, i.e. \*, can be used to accomplish this task.



$ head Babies\_201\*.txt

==> Babies\_2013.txt <==

"Notes" "County" "County Code" "Month" "Month Code" Births

"Baldwin County, AL" "01003" "January" "1" 200

"Baldwin County, AL" "01003" "February" "2" 165

"Baldwin County, AL" "01003" "March" "3" 163

"Baldwin County, AL" "01003" "April" "4" 190

"Baldwin County, AL" "01003" "May" "5" 157

"Baldwin County, AL" "01003" "June" "6" 173

"Baldwin County, AL" "01003" "July" "7" 199

"Baldwin County, AL" "01003" "August" "8" 198

"Baldwin County, AL" "01003" "September" "9" 157

==> Babies\_2014.txt <==

"Notes" "County" "County Code" "Month" "Month Code" Births

"Baldwin County, AL" "01003" "January" "1" 186

"Baldwin County, AL" "01003" "February" "2" 177

"Baldwin County, AL" "01003" "March" "3" 165

"Baldwin County, AL" "01003" "April" "4" 166

"Baldwin County, AL" "01003" "May" "5" 218

"Baldwin County, AL" "01003" "June" "6" 192

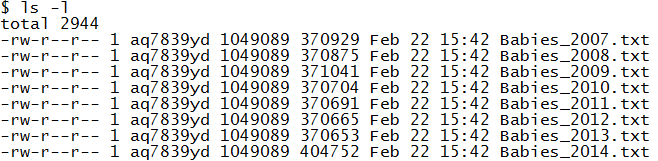
"Baldwin County, AL" "01003" "July" "7" 190

"Baldwin County, AL" "01003" "August" "8" 198

"Baldwin County, AL" "01003" "September" "9" 184

Questions

1. Suppose the following files were contained in my directory. How would one write a single head statement (with wildcards) to show the contents of the first few lines for all these files?



In an effort to investigate the notion of Super Bowl Babies, data was collected on births by month for all counties in the United States. Data from 2007-2014 are provided on our course website. Download each of these files and place them into the same directory as the Babies\_2014.txt and Babies\_2013.txt files.



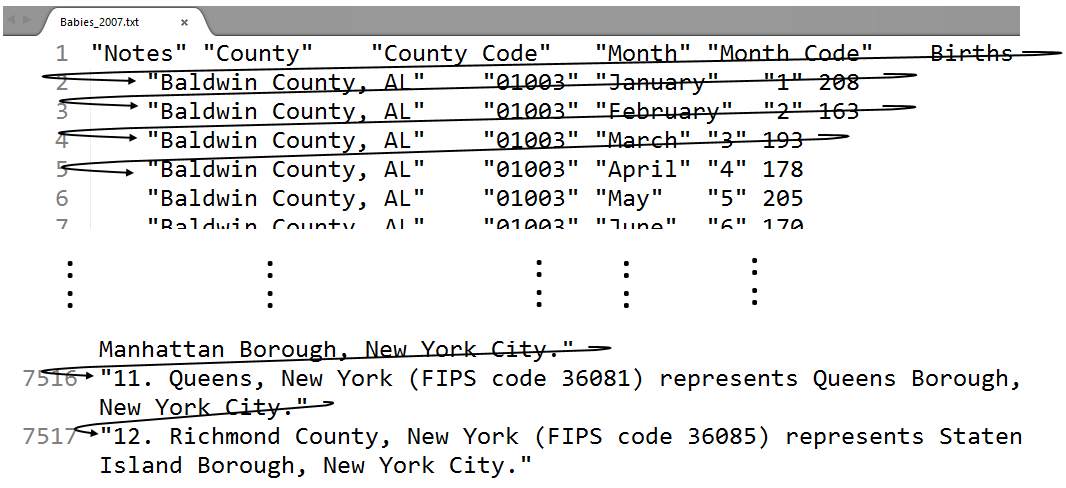
Data Source: <https://wonder.cdc.gov/controller/datarequest/D66>

MANAGEMENT/EDITING OF TEXT WITHIN A FILE

In this section, we will consider the editing of text with in file. There are various BASH utilities that allow for editing text. The common utilities include sed (or gsed on MAC) , awk, and grep.

|  |  |  |
| --- | --- | --- |
| BASH Utility | Full Name |  |
| sed / gsed | Stream Editor | Performs basic text transformations on an input stream |
| awk | AWK Language | Language for pattern scanning and processing |
| grep | Globally search a Regular Expression and Print | Plain-text search utility – regular expression friendly |

The sed utility will be used extensively here as only simple text editing is needed for our work. When the sed command is used, text is continuously feed into the utility line-by-line.



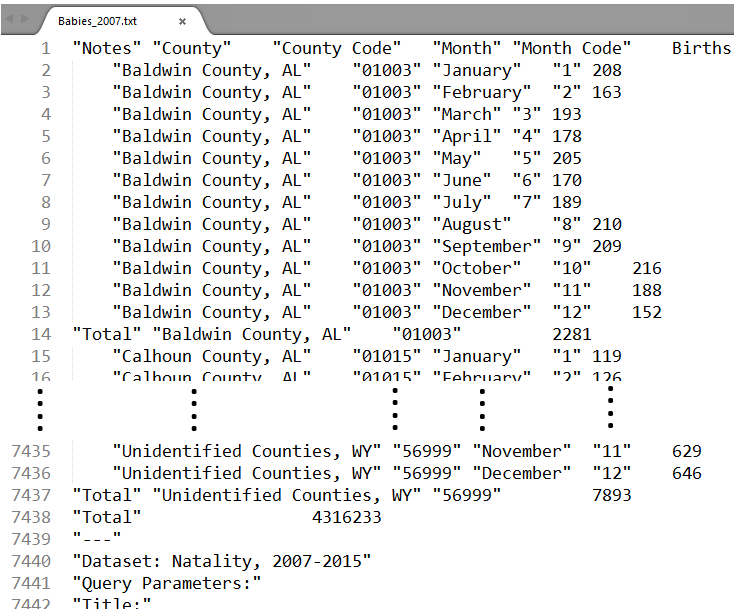
The man command can be used to get help on most BASH commands. You can also search Stack Exchange to get help on using BASH.

Note: The man command is not available in GitBash.

|  |
| --- |
| $ man sed  Usage: sed [OPTION]... {script-only-if-no-other-script} [input-file]...  -n, --quiet, --silent  suppress automatic printing of pattern space  -e script, --expression=script  add the script to the commands to be executed  -f script-file, --file=script-file  add the contents of script-file to the commands to be executed  --follow-symlinks  follow symlinks when processing in place  -i[SUFFIX], --in-place[=SUFFIX]  edit files in place (makes backup if SUFFIX supplied)  -b, --binary  open files in binary mode (CR+LFs are not processed specially)  -l N, --line-length=N  specify the desired line-wrap length for the `l' command  --posix  disable all GNU extensions.  -r, --regexp-extended  use extended regular expressions in the script.  -s, --separate  consider files as separate rather than as a single continuous  long stream.  -u, --unbuffered  load minimal amounts of data from the input files and flush  the output buffers more often  -z, --null-data  separate lines by NUL characters  --help display this help and exit  --version output version information and exit  If no -e, --expression, -f, or --file option is given, then the first  non-option argument is taken as the sed script to interpret. All  remaining arguments are names of input files; if no input files are  specified, then the standard input is read. |

|  |
| --- |
| MAC users will need to use **gsed** whenever a sed command is used throughout. |

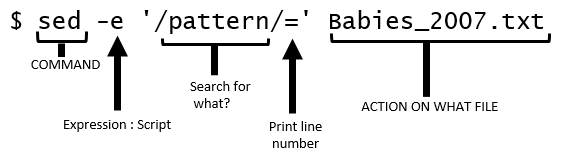
The first step will be to remove the footer information in this file. The footer information starts with the line containing “---“.



The following sed command can used to find the lines that contain “---“.

$ sed -e '/"---"/=' Babies\_2007.txt

Breaking this command into it’s pieces.



The above sed command simply prints the output, i.e. the line number for which “---“ is contained, to the screen. If you’d like to push the output into a file, simple use the following command.

$ sed -e '/"---"/=' Babies\_2007.txt > Babies\_2007\_2.txt

Questions

1. Open Babies\_2007\_2.txt in a text editor. Did the command above print the line numbers for each occurrence of “---“?
2. What does the following command do? Briefly discuss.

$ sed -n '/"---"/=' Babies\_2007.txt

The rm command is used to remove (or delete) a file. Be very careful in using rm with wildcards!

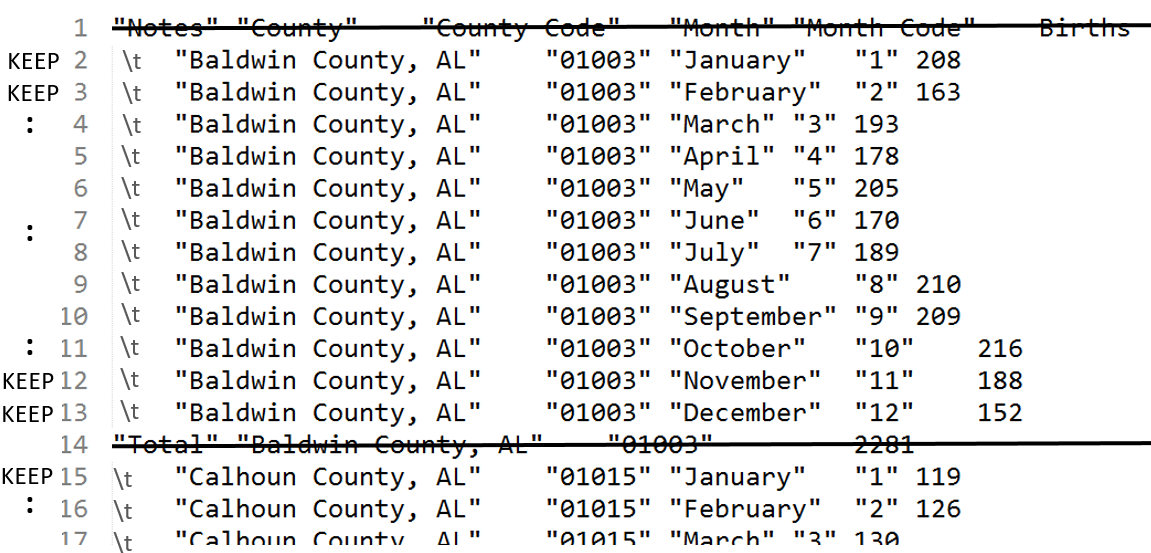
$ rm Babies\_2007\_2.txt

We now know that the footer information in this file begins on line 7439. The relevant content of the file can be obtained using the head command. The first command below simply prints the output (the first 7438 lines) to the screen; whereas, the second version saves the output into a new file called Babies\_2007\_v2.txt.

$ head -7438 Babies\_2007.txt

$ head -7438 Babies\_2007.txt > Babies\_2007\_v2.txt

The next step in the management of these files is to remove the rows that do not contain data for each county. We can accomplish this take in two distinct ways – either delete the non-data rows or keep the data rows. The “keep data rows” approach is shown first.

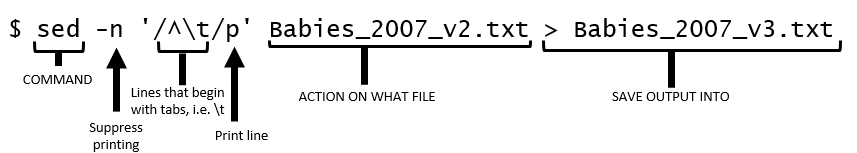


Now, the following command uses sed to print, i.e. keep, only lines that begin with \t. The –n option is used to suppress the printing of content to the screen.

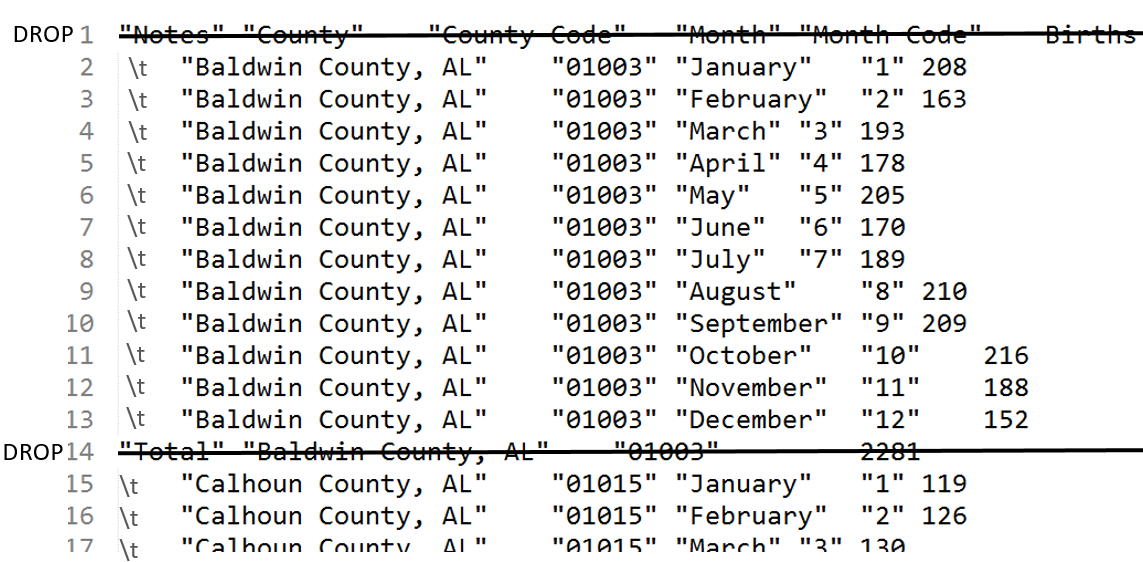
Note: Replace sed with gsed on a MAC to force the use of GNU Bash.

$ sed -n '/^\t/p' Babies\_2007\_v2.txt > Babies\_2007\_v3.txt

The following breaks this command down into its various components.



The following variation of the command above can be used to delete all non-data rows – i.e. rows that start with “ are non-data rows. Rows that contain data start with \t.



$ sed -e '/^”/d' Babies\_2007\_v2.txt > Babies\_2007\_v3.txt

Note: The –n option must be changed to –e here. This is necessary as –n suppresses printing and the letter d (at the end of the quoted string) deletes the output. Thus, an empty file would be returned if –n were not changed to –e.

The last step in the management of these files is to add the year to each line. This can be done using the following substitute functionality which is specified at the beginning of quoted string. Again, gsed should be used in place of sed here on a MAC.

$ sed -e 's/\t/2007\t/' Babies\_2007\_v3.txt > Babies\_2007\_v4.txt

|  |  |
| --- | --- |
| Babies\_2007\_v3.txt | Babies\_2007\_v4.txt |
|  |  |

The following table provides variations of the substitute command.

|  |  |
| --- | --- |
| Substitute Commands | Action |
| sed –e ‘s/old\_text/new\_text/’ <filename> | Replace the 1st instance of old\_text in the line with new\_text |
| sed –e ‘s/old\_text/new\_text/g’ <filename> | Replace all instances of old\_text in the line with new\_text |
| sed –e ‘s/^/new\_text/’ <filename> | Put new\_text at beginning of line |
| sed –e ‘s/$/new\_text/’ <filename> | Put new\_text at end of line |

The -i option can be used to for **in-place editing**. The following command will take the Babies\_2007\_v3.txt file, find the first instance of \t and replace it with 2007\t, and then put the output directly back into Babies\_2007\_v3.txt.

$ sed -i 's/\t/2007\t/' Babies\_2007\_v3.txt

Questions

1. Consider Statement #1 – the command used above. What happens if Statement #2 were used instead? Discuss.

Statement #1: $ sed -e 's/\t/2007\t/' Babies\_2007\_v3.txt

Statement #2: $ sed -e 's/\t/2007/' Babies\_2007\_v3.txt

1. What does the following command do? Discuss.

$ sed -e 's/^/2007/' Babies\_2007\_v3.txt

1. Consider the statement written in the previous problem – identified as Statement #3 here. What adverse effect would Statement #4 have? Discuss.

Statement #3: $ sed -e 's/^/2007/' Babies\_2007\_v3.txt

Statement #4: $ sed -e 's/^/2007\t/' Babies\_2007\_v3.txt

1. Verify that the following command indeed replaces all tab characters, i.e.\t, with 2007.

$ sed -e 's/\t/2007/g' Babies\_2007\_v3.txt

1. What command would be used to place the year at the end of each line. You should carefully consider the placement of any tab and/or new line characters.

The repeated processing of a file can be streamlined using piping.

|  |  |  |
| --- | --- | --- |
| Saving files at each step | Concept of Piping Command | |
|  |  | **Piping**: A sequence of statement chained together so that the output of one process feeds direclty into the next statement. |

Using piping to prepare the Babies\_2007.txt file for appending.

$ head -7438 Babies\_2007.txt | sed -n '/^\t/p' | sed -e 's/\t/2007\t'/ > Babies\_2007\_v2.txt

The following piped commands are used to prepare the datasets from each year.

Note: The footer content of the 2014 files starts on a different line.

|  |  |
| --- | --- |
| Year | Piped Command for Preparing each Year |
| 2007 | $ head -7438 Babies\_2007.txt | sed -n '/^\t/p' | sed -e 's/\t/2007\t/' > Babies\_2007\_v2.txt |
| 2008 | $ head -7438 Babies\_2008.txt | sed -n '/^\t/p' | sed -e 's/\t/2008\t/’ > Babies\_2008\_v2.txt |
| 2009 | $ head -7438 Babies\_2009.txt | sed -n '/^\t/p' | sed -e 's/\t/2009\t/’ > Babies\_2009\_v2.txt |
| 2010 | $ head -7438 Babies\_2010.txt | sed -n '/^\t/p' | sed -e 's/\t/2010\t/' > Babies\_2010\_v2.txt |
| 2011 | $ head -7438 Babies\_2011.txt | sed -n '/^\t/p' | sed -e 's/\t/2011\t/’ > Babies\_2011\_v2.txt |
| 2012 | $ head -7438 Babies\_2012.txt | sed -n '/^\t/p' | sed -e 's/\t/2012\t/’ > Babies\_2012\_v2.txt |
| 2013 | $ head -7438 Babies\_2013.txt | sed -n '/^\t/p' | sed -e 's/\t/2013\t/’ > Babies\_2013\_v2.txt |
| 2014 | $ head -8140 Babies\_2014.txt | sed -n '/^\t/p' | sed -e 's/\t/2014\t/’ > Babies\_2014\_v2.txt |

The following use of head will verify that each file has the correct format. This should be done before appending the files together.

$ head -2 Babies\_20\*\_v2.txt

==> Babies\_2007\_v2.txt <==

2007 "Baldwin County, AL" "01003" "January" "1" 208

2007 "Baldwin County, AL" "01003" "February" "2" 163

==> Babies\_2008\_v2.txt <==

2008 "Baldwin County, AL" "01003" "January" "1" 208

2008 "Baldwin County, AL" "01003" "February" "2" 198

==> Babies\_2009\_v2.txt <==

2009 "Baldwin County, AL" "01003" "January" "1" 187

2009 "Baldwin County, AL" "01003" "February" "2" 145

==> Babies\_2010\_v2.txt <==

2010 "Baldwin County, AL" "01003" "January" "1" 177

2010 "Baldwin County, AL" "01003" "February" "2" 171

==> Babies\_2011\_v2.txt <==

2011 "Baldwin County, AL" "01003" "January" "1" 178

2011 "Baldwin County, AL" "01003" "February" "2" 187

==> Babies\_2012\_v2.txt <==

2012 "Baldwin County, AL" "01003" "January" "1" 153

2012 "Baldwin County, AL" "01003" "February" "2" 162

==> Babies\_2013\_v2.txt <==

2013 "Baldwin County, AL" "01003" "January" "1" 200

2013 "Baldwin County, AL" "01003" "February" "2" 165

==> Babies\_2014\_v2.txt <==

2014 "Baldwin County, AL" "01003" "January" "1" 186

2014 "Baldwin County, AL" "01003" "February" "2" 177

APPENDING FILES in BASH

The next step is to append the files from the individual years into a single file. The cat command can be used to accomplish this task. The full dataset is being placed into a file called Babies\_AllYears.txt.

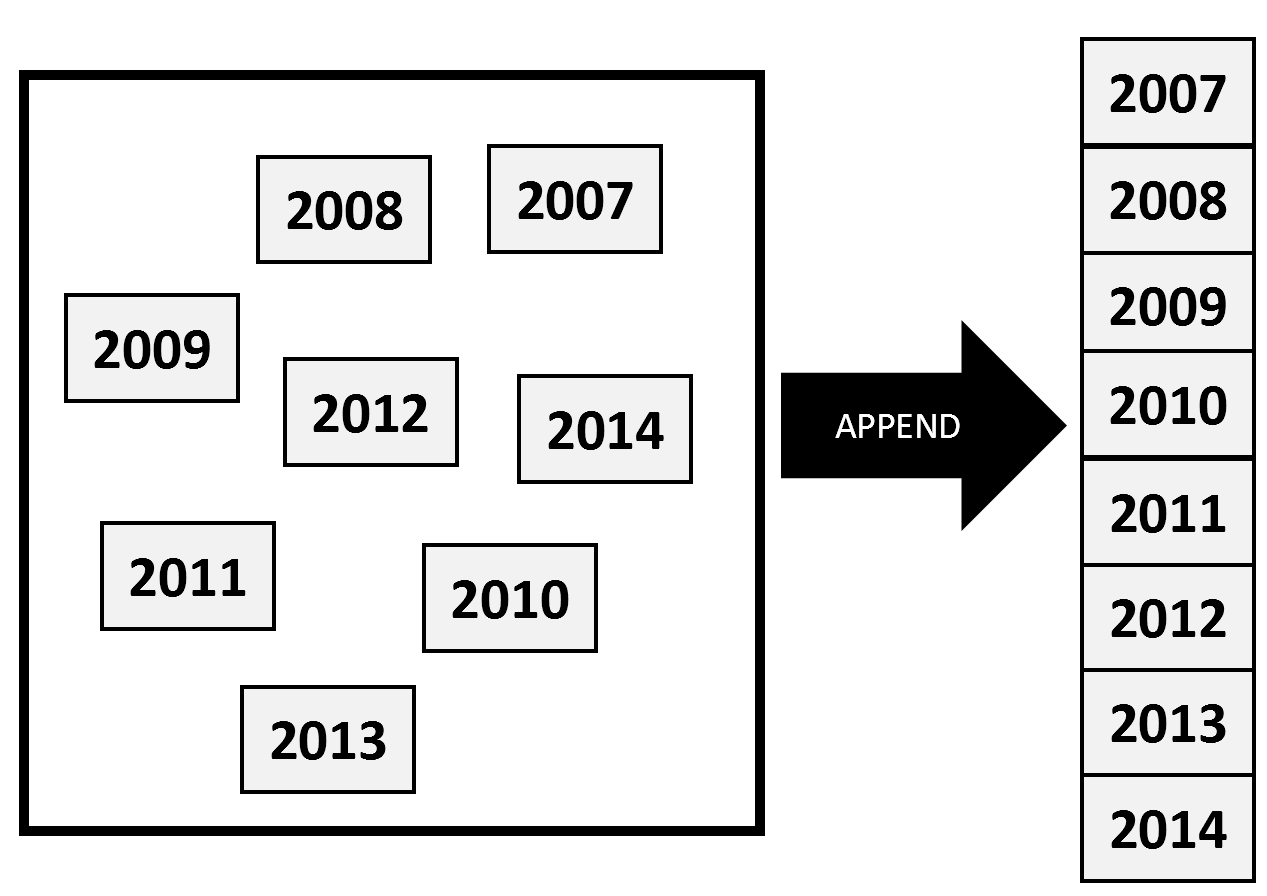
$ cat Babies\_2007\_v2.txt Babies\_2008\_v2.txt Babies\_2009\_v2.txt

Babies\_2010\_v2.txt Babies\_2011\_v2.txt Babies\_2012\_v2.txt

Babies\_2013\_v2.txt Babies\_2014\_v2.txt > Babies\_AllYears.txt

The use of wildcards makes this statement must easier.

$ cat Babies\_20\*\_v2.txt > Babies\_AllYears.txt



The wc command with the –l option will quickly count the number of lines in the file.

$ wc -l Babies\_AllYears.txt

55560 Babies\_AllYears.txt

This count matches the sum of the number of lines from each individual year. A check of this nature provides confidence that appending has worked as intended.

$ wc -l Babies\_20\*\_v2.txt

6864 Babies\_2007\_v2.txt

6864 Babies\_2008\_v2.txt

6864 Babies\_2009\_v2.txt

6864 Babies\_2010\_v2.txt

6864 Babies\_2011\_v2.txt

6864 Babies\_2012\_v2.txt

6864 Babies\_2013\_v2.txt

7512 Babies\_2014\_v2.txt

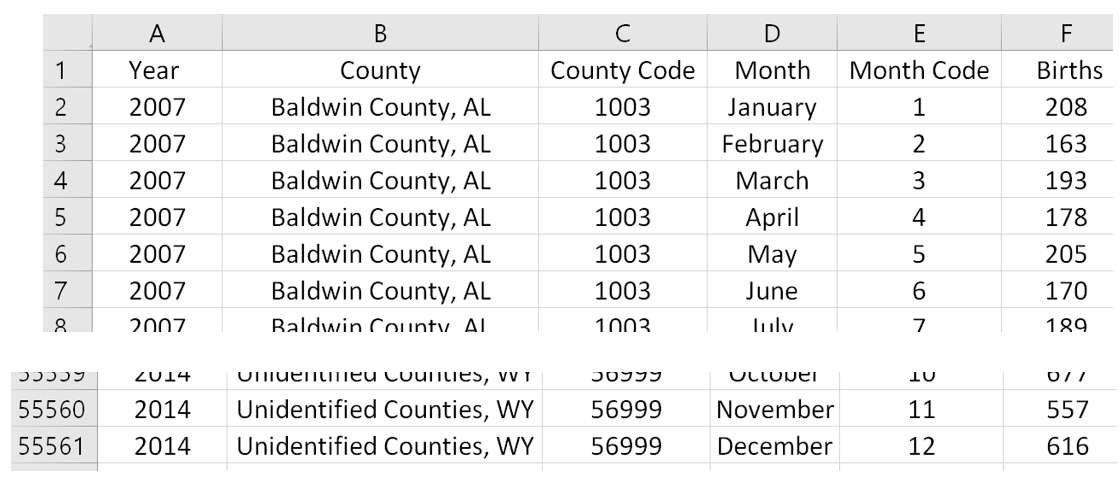
55560 total

ADDING a HEADER

The following sed command (using the –i, i.e. in-place, option) can be used to insert column headers on the first line of this file. The column headers should be tab delimited like the rest of this file so that the headers match the columns of data.

$ sed -i '1 i\Year \t County \t County Code \t Month \t Month Code \t Births' Babies\_AllYears.txt

The preparation of these files is complete. The Babies\_AllYears.txt file can now be read into Excel for further analyses. A snip-it of the Excel file is shown here.



ANALYSIS of DATA

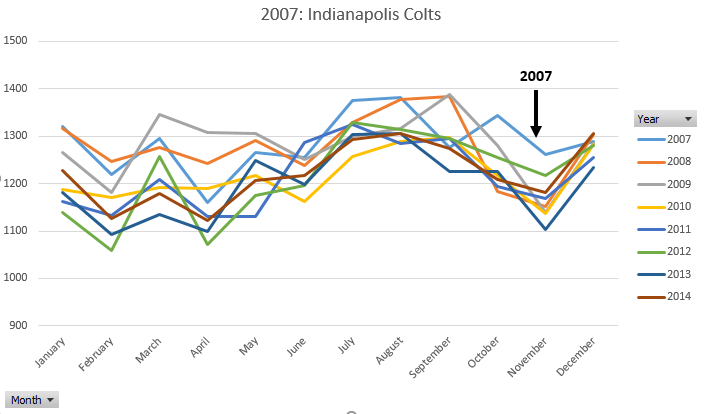
The analysis requires the identification of the Super Bowl winner for each year and the county in which the teams plays. The table below provides a single county for each team. A more thorough analysis could include additional counties for each team.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| List of past Super Bowl Winners    Source: <http://www.topendsports.com/events/super-bowl/winners-list.htm> | Identifying the county for each team   |  |  |  | | --- | --- | --- | | Year | Super Bowl Winner | County | | 2007 | Indianapolis Colts | Marion County, IN | | 2008 | New York Giants | Bergen County, NJ | | 2009 | Pittsburgh Steelers | Allegheny County, PA | | 2010 | New Orleans Saints | Orleans Parish, LA | | 2011 | Green Bay Packers | Brown County, WI | | 2012 | New York Giants | Bergen County, NJ | | 2013 | Baltimore Ravens | Baltimore County, MD | | 2014 | Seattle Seahawks | King County, WA | | 2015 | New England Patriots | Suffolk County, MA | | 2016 | Denver Broncos | Denver County, CO | | 2017 | New England Patriots | Suffolk County, MA | |

Using PivotTables in Excel is the most efficient in getting the necessary summaries. Setup a PivotTable as below. Notice that County is being used as a filter on these summaries. This will allow us to easily summarize the data for each winning Super Bowl team.

|  |  |
| --- | --- |
| Setup of PivotTable | Set Filter = Marion County, IN.    Setting Year Filter = 2007. |

Removing the Year Filter = 2007 allows us to compare 2007 to the typical trends seen from year-to-year in this county.



Consider the following graphs that include outcomes from 2014 (Seattle Seahawks) and 2008 / 2012 (New York Giants).

|  |  |
| --- | --- |
| 2014  Seattle Seahawks  [King County, WA] |  |
| 2008 / 2012  New York Giants  [Bergen County, NJ] |  |

Questions

1. Does the claim of “Super Bowl Babies” appear to be real phenomena? Discuss.
2. Consider our analysis for the Indianapolis Colts (2007). Identify additional counties that surround Marion County, IN. Recreate the graph provided above using these additional counties. Does the same trend appear? Discuss.

TASK

The CDC continually updates their datasets and hence our analyses can be updated using this new data.

Data Sources:

* <https://wonder.cdc.gov/controller/datarequest/D66>
* <https://wonder.cdc.gov/natality-cur_vrent.html>

Complete the following:

1. Download the Babies\_2015.txt file from our course website.
2. Edit this file so that the footer information is removed, the non ^\t rows are removed, and add 2015 to the beginning of each line.
3. Append the 2015 data to the bottom of Babies\_AllYears.txt.
4. Read the Babies\_AllYears.txt file into Excel and update the 2007 Indianapolis Colts graph provided above to include a line for 2015.
5. Create a graph for the 2015 Super Bowl winner the New England Patriots.