8.2 Files

8.2.1 Viewing Files

Using Options

Module 8.1 covered directory navigation, including copying and moving files and viewing the files that are within a directory. This module will cover viewing and editing the contents of files.

First, open Terminal in JupyterHub and change the present working directory (PWD) to `Week.8/8.2.Files`. This directory contains a file named `cancer_pts.txt`.

```
j:~$ cd Week.8/8.2.Files
j:~/Week.8/8.2.Files$ ls
cancer_pts.txt  Supplement.8.2.pdf  Tutorial.8.2
j:~/Week.8/8.2.Files$
```

There are many commands for viewing file contents in the command line. As files can be quite large, especially files containing experimental results, it is often useful to look at just the first few lines. The `head` command outputs the first 10 lines of a file.

```
head [OPTION]... [FILE]...
```

`FILE` is the absolute or relative path to the file. View the first 10 lines of the file `cancer_pts.txt` file with the `head` command:

```
j:~/Week.8/8.2.Files$ head cancer_pts.txt
patient.ID      sex     age     cancer.type
DO49528 Male    54      Acute myeloid leukemia
DO49529 Male    54      Acute myeloid leukemia
DO49531 Male    49      Acute myeloid leukemia
DO49532 Female  39      Acute myeloid leukemia
DO49533 Male    39      Acute myeloid leukemia
DO49536 Male    46      Acute myeloid leukemia
DO49537 Female  50      Acute myeloid leukemia
DO52732 Female  75      Acute myeloid leukemia
DO52739 Female  66      Acute myeloid leukemia
j:~/Week.8/8.2.Files$
```

The file `cancer_pts.txt` is a tab delimited file. The first line of the file is a header line describing the 4 columns: patient.ID, sex, age, and cancer type. The following 9 lines contain information for 9 patients.

The usage syntax above shows that the `head` command can take options. To learn the options for a command, any command can be run with the option `--help`.

```
j:~/Week.8/8.2.Files$ head --help
Usage: head [OPTION]... [FILE]...
Print the first 10 lines of each FILE to standard output.
With more than one FILE, precede each with a header giving the file name.

With no FILE, or when FILE is -, read standard input.

Mandatory arguments to long options are mandatory for short options too.
  -c, --bytes=[-]NUM      print the first NUM bytes of each file;
                            with the leading '-', print all but the last
                            NUM bytes of each file
  -n, --lines=[-]NUM      print the first NUM lines instead of the first 10;
                            with the leading '-', print all but the last
                            NUM lines of each file
  -q, --quiet, --silent   never print headers giving file names
  -v, --verbose           always print headers giving file names
  -z, --zero-terminated   line delimiter is NUL, not newline
      --help     display this help and exit
      --version  output version information and exit

NUM may have a multiplier suffix:
b 512, kB 1000, K 1024, MB 1000*1000, M 1024*1024,
GB 1000*1000*1000, G 1024*1024*1024, and so on for T, P, E, Z, Y.

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation at: <https://www.gnu.org/software/coreutils/head>
or available locally via: info '(coreutils) head invocation'
j:~/Week.8/8.2.Files$
```

The help page contains the following information:

- Usage (how to use the command)
- A description of what the command does
- Options that can be used with the command
- Any other extra information about the command

In the options section, each option is described. Here is one of the options:

```
-n, --lines=[-]NUM  print the first NUM lines instead of the first 10;
                        with the leading '-', print all but the last
                        NUM lines of each file
```

Some options have two versions, a **short option** and a **long option**. Short options are a single letter and are preceded by a single hyphen, long options are multiple letters and are preceded by two hyphens. In this case the short option is `-n` and the long option is `--lines`. Short and long options can be used interchangeably.

Here is another one of the options:

```
-v, --verbose        always print headers giving file names
```

In this case the short option is `-v` and the long option is `--verbose`. Unlike the `-n` option, there is no equals sign after the long option for the `-v` option. This is because the `-n` option takes an argument and the `-v` option does not take an argument. The `-n` option is used to

tell the command how many lines to output, so an argument (the number of lines) is required. Alternatively, the `-v` option tells the command to print headers, which doesn't require any more information in the form of an argument.

Options without arguments are run by typing the either the long or short option after the command name. The `-v` option changes the output of the head command by printing a header with the file name. For example, see how it alters the output with the file `cancer_pts.txt`:

```
j:~/Week.8/8.2.Files$ head -v cancer_pts.txt
==> cancer_pts.txt <==
patient.ID      sex     age     cancer.type
DO49528 Male    54      Acute myeloid leukemia
DO49529 Male    54      Acute myeloid leukemia
DO49531 Male    49      Acute myeloid leukemia
DO49532 Female  39      Acute myeloid leukemia
DO49533 Male    39      Acute myeloid leukemia
DO49536 Male    46      Acute myeloid leukemia
DO49537 Female  50      Acute myeloid leukemia
DO52732 Female  75      Acute myeloid leukemia
DO52739 Female  66      Acute myeloid leukemia
j:~/Week.8/8.2.Files$
```

The output is different from the previous use of the head command in that the name of the file is printed before the first 10 lines of `cancer_pts.txt`.

Long and short options can be used interchangeably. Using `--verbose` instead of `-v` will produce the same result:

```
j:~/Week.8/8.2.Files$ head --verbose cancer_pts.txt
==> cancer_pts.txt <==
patient.ID      sex     age     cancer.type
DO49528 Male    54      Acute myeloid leukemia
DO49529 Male    54      Acute myeloid leukemia
DO49531 Male    49      Acute myeloid leukemia
DO49532 Female  39      Acute myeloid leukemia
DO49533 Male    39      Acute myeloid leukemia
DO49536 Male    46      Acute myeloid leukemia
DO49537 Female  50      Acute myeloid leukemia
DO52732 Female  75      Acute myeloid leukemia
DO52739 Female  66      Acute myeloid leukemia
j:~/Week.8/8.2.Files$
```

The `-n` option changes the number of lines output by the `head` command based on an argument provided to the option `NUM`:

```
-n, --lines=[-]NUM  print the first NUM lines instead of the first 10;
                        with the leading '-', print all but the last
                        NUM lines of each file
```

When provided an argument with an option it differs depending on whether the short or long option is used.

Short option: `-o ARG`
Long option: `--option=ARG`

To output the first three lines of the `cancer_pts.txt` file, either `-n 3` or `-lines=3` can be used. Both are shown below and produce the exact same output.

```
j:~/Week.8/8.2.Files$ head -n 3 cancer_pts.txt
patient.ID      sex     age     cancer.type
DO49528 Male    54      Acute myeloid leukemia
DO49529 Male    54      Acute myeloid leukemia
j:~/Week.8/8.2.Files$ head -lines=3 cancer_pts.txt
patient.ID      sex     age     cancer.type
DO49528 Male    54      Acute myeloid leukemia
DO49529 Male    54      Acute myeloid leukemia
j:~/Week.8/8.2.Files$
```

Multiple options can be used together. For example, to output the first three lines of `cancer_pts.txt` along with the file name, the `-n` and `-v` options can be used together. The order in which the options are applied does not matter.

```
j:~/Week.8/8.2.Files$ head -n 3 -v cancer_pts.txt
==> cancer_pts.txt <==
patient.ID      sex     age     cancer.type
DO49528 Male    54      Acute myeloid leukemia
DO49529 Male    54      Acute myeloid leukemia
j:~/Week.8/8.2.Files$ head -v -n 3 cancer_pts.txt
==> cancer_pts.txt <==
patient.ID      sex     age     cancer.type
DO49528 Male    54      Acute myeloid leukemia
DO49529 Male    54      Acute myeloid leukemia
j:~/Week.8/8.2.Files$
```

However, an option and its argument must be typed together. If the -v option is used between the -n option and the argument (3), an error message will be output, and the command will not work.

```
j:~/Week.8/8.2.Files$ head -n -v 3 cancer_pts.txt
head: invalid number of lines: ‘v’
j:~/Week.8/8.2.Files$
```

File Viewing Commands

Multiple other commands exist for viewing file contents. The `tail` command outputs the LAST 10 lines of a file.

```
tail [OPTION]... [FILE]...
```

The `tail` command has the same options as the `head` command. For example, to see the last 3 lines of `cancer_pts.txt` the `-n` option can be used:

```
j:~/Week.8/8.2.Files$ tail -n 3 cancer_pts.txt
DO43514 Female  71      Uterine Endometrioid Carcinoma
DO43610 Female  82      Uterine Endometrioid Carcinoma
DO43739 Female  59      Uterine Endometrioid Carcinoma
j:~/Week.8/8.2.Files$
```

The `head` and `tail` commands have the `-c` or `--bytes` option which, depending on the command, outputs the first or last `NUM` bytes of the file. Each letter in a file is a byte. To output the last 5 bytes of `cancer_pts.txt` the `-c` option can be used with 5 as the argument:

```
j:~/Week.8/8.2.Files$ tail -c 5 cancer_pts.txt
noma
j:~/Week.8/8.2.Files$
```

Only 4 letters were output. Looking back at the output of `tail -n 3 cancer_pts.txt` above, these are the last 4 letters in the last line of the file. The reason only 4 letters are output is because the end of a line is also counted as one byte. To get the last 5 letters, the number 6 should be used as the argument for `-c` as it accounts for the line ending byte.

The `cat` command outputs an entire file.

```
cat [OPTION]... [FILE]...
```

An example of `cat` will be shown in 8.2.2.

The `less` command allows one to view the entire contents of a file without outputting the contents to the Terminal. To understand how this command works, run the command:

```
less cancer_pts.txt
```

The file contents will take up the whole screen. Using up and down arrows the file contents can be scrolled through. To return to the command line, press the letter Q. The less command tends is convenient for large files as it will not output the entire file to the terminal.

8.2.2 Redirection

Redirecting Standard Output to Files

Standard output (stdout) is the data produced by a command, or in other words, the data that is returned after a command is run. If a command produces standard output, it is displayed in the terminal. The `ls` command produces standard output. When the `ls` command is executed, a list of files and directories are output in the Terminal on the lines

following the command and before the next command prompt. The `cd` command, however, does not produce standard output. When the `cd` command is executed, the next line will display a command prompt.

```
j:~/Week.8/8.2.Files$ ls
cancer_pts.txt  Supplement.8.2.pdf  Tutorial.8.2
j:~/Week.8/8.2.Files$ cd ..
j:~/Week.8$ cd 8.2.Files
j:~/Week.8/8.2.Files$
```

Other commands that produce standard output include: `head`, `tail`, `cat`, & `pwd`. Other commands that do not produce standard output include: `cp`, `mv`, & `rm`.

Instead of having the standard output displayed in the terminal, it can be **redirected** to a file or to another command. Redirection alters the destination of the standard output. To redirect standard output to a file instead of displaying it in the terminal, the output redirection operator is used. The output redirection operator is a greater than symbol: >. To save the output of a command to a file:

```
command > file
```

For example, the output redirection operator can be used to create a file called `first_pts.txt` that contains the information for the first 5 patients in the `cancer_pts.txt` file and the header row. To get the header row and the lines for the 5 patients, the `head` command can be used with `-n 6`. After the head command the redirection operator can be typed followed by the name of the file.

```
j:~/Week.8/8.2.Files$ head -n 6 cancer_pts.txt > first_pts.txt
j:~/Week.8/8.2.Files$ cat first_pts.txt
patient.ID      sex     age     cancer.type
DO49528 Male    54      Acute myeloid leukemia
DO49529 Male    54      Acute myeloid leukemia
DO49531 Male    49      Acute myeloid leukemia
DO49532 Female  39      Acute myeloid leukemia
DO49533 Male    39      Acute myeloid leukemia
j:~/Week.8/8.2.Files$
```

Using the `cat` command to view the full contents of `first_pts.txt`, it contains the first 6 lines of `cancer_pts.txt`.

Before the output of the `head` command was redirected to `first_pts.txt` there was no file with that name. Now the file does exist, if data is redirected to this file, it will overwrite the existing contents. For example, if the last two lines of `cancer_pts.txt` are redirected to `first_pts.txt` the file will only contain those two lines.

```
j:~/Week.8/8.2.Files$ tail -n 2 cancer_pts.txt > first_pts.txt
j:~/Week.8/8.2.Files$ cat first_pts.txt
DO43610 Female  82      Uterine Endometrioid Carcinoma
DO43739 Female  59      Uterine Endometrioid Carcinoma
j:~/Week.8/8.2.Files$
```

To add to a file instead of overwriting the contents of the file, two greater than symbols are used:

```
command >> file
```

For example, `first_pts.txt` current contains the last two lines of `cancer_pts.txt`. To add the first three lines of `cancer_pts.txt` to the lines that are already in `first_pts.txt` the >> operator can be used:

```
j:~/Week.8/8.2.Files$ head -n 3 cancer_pts.txt >> first_pts.txt
j:~/Week.8/8.2.Files$ cat first_pts.txt
DO43610 Female  82      Uterine Endometrioid Carcinoma
DO43739 Female  59      Uterine Endometrioid Carcinoma
patient.ID      sex     age     cancer.type
DO49528 Male    54      Acute myeloid leukemia
DO49529 Male    54      Acute myeloid leukemia
j:~/Week.8/8.2.Files$
```

Note the lines added with the >> operator are added to the end of the file, after the existing contents.

Redirecting Standard Output to Commands

Standard output can be redirected to files or to other commands. To redirect standard output to another command instead of displaying it in the terminal, the pipe operator is used. This works like the dpylr pipe operator (`%>%`) from module 5.3. The pipe operator in the command line is a vertical bar (like "or" in R): `|`. To pipe the output of a command to another command:

```
command | command
```

Output from the first command will be used as the required argument to the second command. For example, to output the first item listed in a directory, `head -n 1` can be applied to the results of the `ls` command.

```
j:~/Week.8/8.2.Files$ ls
cancer_pts.txt  Supplement.8.2.pdf  Tutorial.8.2
j:~/Week.8/8.2.Files$ ls | head -n 1
cancer_pts.txt
j:~/Week.8/8.2.Files$
```

The first item in the result of `ls` is `cancer_pts.txt`. The `head -n 1` command uses the result of `ls` as the input instead of a file.

Some commands cannot use data provided via piping standard output. For example, the `mkdir` command, which has the usage: `mkdir [OPTION]... DIRECTORY...`

Alternatively, the head command, which *can* use data provided via piping standard output (as above), has the usage: `head [OPTION]... [FILE]...`

The square brackets around the argument for the `head` command (`[FILE]`) indicate that the argument can be provided via standard output. The lack of square brackets in the `mkdir` command (`DIRECTORY`) indicate that the argument **cannot** be provided via standard output.

The `tail` command can also use data from standard output for its argument. For example, to get only the 4th line of the `cancer_pts.txt` file, one could output the first 4 lines of the file, and the get the last of the 4 lines. This can be achieved by piping the output of `head -n 4 cancer_pts.txt` to `tail -n 1`.

```
j:~/Week.8/8.2.Files$ head -n 4 cancer_pts.txt
patient.ID      sex     age     cancer.type
DO49528 Male    54      Acute myeloid leukemia
DO49529 Male    54      Acute myeloid leukemia
DO49531 Male    49      Acute myeloid leukemia
j:~/Week.8/8.2.Files$ head -n 4 cancer_pts.txt | tail -n 1
DO49531 Male    49      Acute myeloid leukemia
j:~/Week.8/8.2.Files$
```

The output of the command is the last line of the first 4 lines of the file. More examples of piping and more information about piping commands is in 8.2.3.

8.2.3 Editing Files

Cut & Paste

The `cancer_pts.txt` file is tab delimited. It has 4 columns: patient.ID, sex, age, & cancer.type. These are also called **fields**. The first field in the file is patient.ID, and the third field is age.

```
J:~/Week.8/8.2.Files$ head cancer_pts.txt
patient.ID      sex     age     cancer.type
DO49528 Male    54      Acute myeloid leukemia
DO49529 Male    54      Acute myeloid leukemia
DO49531 Male    49      Acute myeloid leukemia
DO49532 Female  39      Acute myeloid leukemia
DO49533 Male    39      Acute myeloid leukemia
DO49536 Male    46      Acute myeloid leukemia
DO49537 Female  50      Acute myeloid leukemia
DO52732 Female  75      Acute myeloid leukemia
DO52739 Female  66      Acute myeloid leukemia
j:~/Week.8/8.2.Files$
```

It's important to note that fields in tab delimited format will often not line up in different lines of the file. Above, the patient sex in lines 2-10 does not **visually** line up with the word "sex" in the header row, although "sex" is still the header associated with that column. This

is because the length of the value in each field determines the location of the next tab. Keep this in mind when looking at data; if you count the number of headers, it may be easier to determine which column they belong to.

The `cut` command extracts selected parts of each file.

```
cut OPTION... [FILE]...
```

Note that in the usage for the `cut` command, `OPTION` is not in square brackets. This is because at least one option must be used with the `cut` command. Options for the command include:

```
  -b, --bytes=LIST        select only these bytes
  -c, --characters=LIST   select only these characters
  -d, --delimiter=DELIM   use DELIM instead of TAB for field delimiter
  -f, --fields=LIST       select only these fields; also print any line
                            that contains no delimiter character,
                            unless the -s option is specified
```

The help page of the `cut` command includes the following information on how to use the command:

```
Use one, and only one of -b, -c or -f.  Each LIST is made up of one
range, or many ranges separated by commas.  Selected input is written
in the same order that it is read, and is written exactly once.
Each range is one of:

  N     N'th byte, character or field, counted from 1
  N-    from N'th byte, character or field, to end of line
  N-M   from N'th to M'th (included) byte, character or field
  -M    from first to M'th (included) byte, character or field
```

In other words, the user must choose to extract bytes (`-b`), characters (`-c`), or fields (`-f`) from each line in a file and provide a list of the bytes, characters, or fields to extract using one of the list formats above.

For example, the `cut` command can be used to extract the first and third columns of `cancer_pts.txt` and output the result to a file called `pt_ages.txt`. By default, the `cut` command assumes the fields are separated by tabs. The –delimiter (-d) option can be used if this is not the case.

```
j:~/Week.8/8.2.Files$ cut -f 1,3 cancer_pts.txt > pt_ages.txt
j:~/Week.8/8.2.Files$ head -n 3 pt_ages.txt
patient.ID      age
DO49528 54
DO49529 54
j:~/Week.8/8.2.Files$
```

Viewing the output of `pt_ages.txt`, it contains only the patient.ID (first) and age (third) fields of `cancer_pts.txt`. Similarly, the second column/field of `cancer_pts.txt` can be saved to a file named `pt_sex.txt`.

```
j:~/Week.8/8.2.Files$ cut -f 2 cancer_pts.txt > pt_sex.txt
j:~/Week.8/8.2.Files$ head -n 3 pt_sex.txt
sex
Male
Male
j:~/Week.8/8.2.Files$
```

The `paste` command pastes together corresponding lines of two or more files, separated by tabs.

```
paste [OPTION]... [FILE]...
```

For example, the contents of the `pt_ages.txt` and `pt_sex.txt` can be pasted together. The order in which the files are provided to the `paste` command determines the output.

```
j:~/Week.8/8.2.Files$ paste pt_ages.txt pt_sex.txt | head -n 3
patient.ID      age     sex
DO49528 54      Male
DO49529 54      Male
j:~/Week.8/8.2.Files$ paste pt_sex.txt pt_ages.txt | head -n 3
sex     patient.ID      age
Male    DO49528 54
Male    DO49529 54
j:~/Week.8/8.2.Files$
```

The `paste` command has an option `-d`:

```
  -d, --delimiters=LIST  reuse characters from LIST instead of TABs
```

Instead of pasting files next to one another separated by a tab, a comma delimiter can be used instead:

```
j:~/Week.8/8.2.Files$ paste -d , pt_ages.txt pt_sex.txt | head -n 3
patient.ID      age,sex
DO49528 54,Male
DO49529 54,Male
j:~/Week.8/8.2.Files$
```

Because the `paste` command can take multiple arguments, the syntax is different if data is piped to the `paste` command. A hyphen is used to tell the `paste` command which argument is being replaced by the standard output.

`command | paste file -` OR `command | paste - file`

For example, if the cancer.type field (4th field) from `cancer_pts.txt` directly extracted using the `cut` command and pasted to the `pt_sex.txt` file the command should be written as follows:

```
j:~/Week.8/8.2.Files$ cut -f 4 cancer_pts.txt | paste pt_sex.txt - |
head -n 3
sex     cancer.type
Male    Acute myeloid leukemia
Male    Acute myeloid leukemia
j:~/Week.8/8.2.Files$ cut -f 4 cancer_pts.txt | paste - pt_sex.txt |
head -n 3
cancer.type     sex
Acute myeloid leukemia  Male
Acute myeloid leukemia  Male
j:~/Week.8/8.2.Files$
```

This technique can also be used for any other commands that use multiple arguments.

Editing with sed

The `sed` command performs basic text editing.

```
sed [OPTION]... script [FILE]...
```

The `sed` command requires a script, which is provided in single quotation marks. Some common and useful sed scripts include:

Substitute pattern with replacement: `'s/pattern/replacement/g'`
Delete Nth line: `'Nd'`
Insert line before Nth line: `'Ni text'`

The most common use for sed is substituting a pattern with a replacement, much like the `gsub()` function in R. For example, replacing the word "Male" with the letter "M" in `cancer_pts.txt`. "Male" is used as the pattern and "M" as the replacement in the `sed` script:

```
j:~/Week.8/8.2.Files$ sed 's/Male/M/g' cancer_pts.txt | head -n 6
patient.ID      sex     age     cancer.type
DO49528 M       54      Acute myeloid leukemia
DO49529 M       54      Acute myeloid leukemia
DO49531 M       49      Acute myeloid leukemia
DO49532 Female  39      Acute myeloid leukemia
DO49533 M    39      Acute myeloid leukemia
j:~/Week.8/8.2.Files$
```

In the output, the word "Male" has been replaced by the letter "M". Note that in the 5th row of the output "male" in "Female" has not been replaced by the letter "M". The lowercase "male" in "Female" does not match the pattern and is therefore not replaced.

To insert a new row in output of `sed 's/Male/M/g'` cancer_pts.txt the `sed` insertion script can be used. Below, a line is inserted before the 2nd line containing the text "a new row".

```
j:~/Week.8/8.2.Files$ sed 's/Male/M/g' cancer_pts.txt | sed '2i a new
row' | head -n 6
patient.ID      sex     age     cancer.type
a new row
DO49528 M       54      Acute myeloid leukemia
DO49529 M       54      Acute myeloid leukemia
DO49531 M       49      Acute myeloid leukemia
DO49532 Female  39      Acute myeloid leukemia
j:~/Week.8/8.2.Files$
```

The `sed` substitution script can be used to make a CSV version of the tab delimited `cancer_pts.txt`. To represent a tab character in the command line, the tab escape character is used: `\t`. The command below converts all tabs to commas in `cancer_pts.txt` and outputs the result to a new file named `cancer_pts.csv`.

```
j:~/Week.8/8.2.Files$ sed 's/\t/,/g' cancer_pts.txt > cancer_pts.csv
j:~/Week.8/8.2.Files$ head -n 3 cancer_pts.csv
patient.ID,sex,age,cancer.type
DO49528,Male,54,Acute myeloid leukemia
DO49529,Male,54,Acute myeloid leukemia
j:~/Week.8/8.2.Files$
```

The cut command has an option that permits it to be used with CSV files:

```
  -d, --delimiter=DELIM  use DELIM instead of TAB for field delimiter
```

To extract the 2nd to the 4th fields of `cancer_pts.csv` the `-d` option of the `cut` command can be applied to change the delimiter from tabs to commas.

```
j:~/Week.8/8.2.Files$ cut -d , -f 2-4 cancer_pts.csv | head -n 3
sex,age,cancer.type
Male,54,Acute myeloid leukemia
Male,54,Acute myeloid leukemia
j:~/Week.8/8.2.Files$
```

If used without the `-d` option, the `cut` command cannot be applied to CSV files:

```
j:~/Week.8/8.2.Files$ cut -f 3 cancer_pts.csv | head -n 3
patient.ID,sex,age,cancer.type
DO49528,Male,54,Acute myeloid leukemia
DO49529,Male,54,Acute myeloid leukemia
j:~/Week.8/8.2.Files$
```

The entirety of each line is returned instead of the 3rd field.

Another useful `cut` command option is the `-c` option. This option is used to specify which characters to return instead of selecting fields. The following commands, in order, output:

- the 5th to the 10th character on each line

- all the characters up to (and including) the 7th character
- all the characters from the 25th character to the end of the line (including the 25th character)

```
j:~/Week.8/8.2.Files$ cut -c 5-10 cancer_pts.csv | head -n 3
ent.ID
528,Ma
529,Ma
j:~/Week.8/8.2.Files$ cut -c -7 cancer_pts.csv | head -n 3
patient
DO49528
DO49529
j:~/Week.8/8.2.Files$ cut -c 25- cancer_pts.csv | head -n 3
r.type
eloid leukemia
eloid leukemia
j:~/Week.8/8.2.Files$
```