

BINF2111 – Introduction to Bioinformatics Computing

UNIX 101 part trois (grep regex and sed)



**Richard Allen White III, PhD
RAW Lab**

Lecture 4 - Thursday Aug 31st, 2023

Learning Objectives

- Review quiz and bonus
- Regular expressions in grep
- Sed
- Regular expressions in sed
- Quiz 4

grep vs. python vs. rust

```
>seq1
ATACTCATATCAT
>seq2
ATCTGCTCGCTAG
>seq3
AGCTCGATCAGCG
```

Linux terminal (bash) commands:

```
grep '>' one.fasta | wc -l
Or:
grep -c '>' one.fasta
```

Python Script:

```
#!/usr/bin/env python

import sys

count = 0
with open(sys.argv[1]) as reader:
    for line in reader:
        if line.startswith('>'):
            count += 1
print(count)
```

Rust

```
1 use std::env;
2 use std::fs::File;
3 use std::io::{BufReader, Read};
4
5 fn main() -> Result<(), Box<dyn std::error::Error>> {
6     // Get the command-line arguments
7     let args: Vec<String> = env::args().collect();
8
9     // Check if an argument (FASTA file path) is provided
10    if args.len() != 2 {
11        eprintln!("Usage: {} <fasta_file>", args[0]);
12        std::process::exit(1);
13    }
14
15    let filename = &args[1];
16    let file = File::open(filename)?;
17
18    let mut reader = BufReader::new(file);
19    let mut buffer = [0u8; 8192]; // Read in 8KB chunks
20
21    let mut count = 0;
22    let mut in_header = false;
23
24    loop {
25        let bytes_read = reader.read(&mut buffer)?;
26        if bytes_read == 0 {
27            break;
28        }
29
30        for &byte in buffer.iter().take(bytes_read) {
31            match byte {
32                b'>' => {
33                    if !in_header {
34                        count += 1;
35                        in_header = true;
36                    }
37                }
38                b'\n' => {
39                    in_header = false;
40                }
41                _ => {}
42            }
43        }
44    }
45
46    println!("Number of '>' symbols: {}", count);
47
48    Ok(())
49 }
```

grep vs. python vs. rust

```
Number of '>' symbols: 3
```

```
(base) drrichardallenwhiteiii@system76-pc:~/fasta_counter$ time ./fastacounter example.fasta
```

```
Number of '>' symbols: 3
```

```
real    0m0.002s
```

```
user    0m0.002s
```

```
sys     0m0.000s
```

```
(base) drrichardallenwhiteiii@system76-pc:~/fasta_counter$ time grep ">" -c example.fasta
```

```
3
```

```
real    0m0.002s
```

```
user    0m0.002s
```

```
sys     0m0.000s
```

```
(base) drrichardallenwhiteiii@system76-pc:~/fasta_counter$ time python count.py example.fasta
```

```
3
```

```
real    0m0.020s
```

```
user    0m0.020s
```

```
sys     0m0.000s
```

```
(base) \> time rustc src/main.rs --target x86_64-unknown-linux-gnu --release --out-dir target/release -- --example example.fasta
```

grep vs. python vs. rust

```
sys      0m0.000s
(base) drrichardallenwhiteiii@system76-pc:~/fasta_counter$ time python count.py example.fasta
3

real     0m0.020s
user     0m0.020s
sys      0m0.000s
(base) drrichardallenwhiteiii@system76-pc:~/fasta_counter$ time python count.py example.fasta
3

real     0m0.019s
user     0m0.015s
sys      0m0.004s
(base) drrichardallenwhiteiii@system76-pc:~/fasta_counter$ time python count.py example.fasta
3

real     0m0.019s
user     0m0.018s
sys      0m0.000s
(base) drrichardallenwhiteiii@system76-pc:~/fasta_counter$ time python count.py example.fasta
3

real     0m0.019s
user     0m0.015s
sys      0m0.004s
(base) drrichardallenwhiteiii@system76-pc:~/fasta_counter$ time python count.py example.fasta
3

real     0m0.019s
user     0m0.018s
sys      0m0.000s
```

grep vs. python vs. rust

```
sys      0m0.000s
(base) drrichardallenwhiteiii@system76-pc:~/fasta_counter$ time ./fastacounter example.fasta
Number of '>' symbols: 3

real     0m0.002s
user     0m0.002s
sys      0m0.000s
(base) drrichardallenwhiteiii@system76-pc:~/fasta_counter$ time ./fastacounter example.fasta
Number of '>' symbols: 3

real     0m0.002s
user     0m0.002s
sys      0m0.000s
(base) drrichardallenwhiteiii@system76-pc:~/fasta_counter$ time ./fastacounter example.fasta
Number of '>' symbols: 3

real     0m0.002s
user     0m0.002s
sys      0m0.000s
(base) drrichardallenwhiteiii@system76-pc:~/fasta_counter$
```

Quiz 3 answers

Command 1

```
echo "ATATCCC" | grep -c "AT"
```

Command 2

```
echo "ATATCCC" | grep "AT" | wc -l
```

How many AT repeats does it count?

Quiz 3 answers

Command 1

```
echo "ATATCCC" | grep -c "AT"
```

Command 2

```
echo "ATATCCC" | grep "AT" | wc -l
```

How many AT repeats does it count?

One each

Quiz 3 answers

Command 1

```
echo "ATATCCC" | grep -c "AT"
```

Command 2

```
echo "ATATCCC" | grep "AT" | wc -l
```

How many AT repeats does it count?

How would I count both AT's?

Quiz 3 answers

How would I count both AT's?

Command 1

```
echo "ATATCCC" | grep -o "AT" | wc -l
```

Is there another way?

Quiz 3 answers

How would I count both AT's?

Command 1

```
echo "ATATCCC" | grep -o "AT" | wc -l
```

Is there another way?

Command 2

```
echo "ATATCCC" | grep -o "AT" | grep -c ""
```

Another way?

Quiz 3 answers

How would I count both AT's?

```
echo "ATATCCC" | grep -o "AT" | wc -l
```

```
echo "ATATCCC" | grep -o "AT" | grep -c ""
```

Another way?

```
echo "ATATCC" | grep -o "AT\|AT" | grep -c "AT"
```

or

Quiz 3 answers

How would I count both AT's?

```
echo "ATATCCC" | grep -o "AT" | wc -l
```

```
echo "ATATCCC" | grep -o "AT" | grep -c ""
```

Another way?

```
echo "ATATCC" | grep -o "AT\|AT" | grep -c "AT"
```

or

```
echo -n "ATATCC" | tr -d '\n' | grep -o "AT" | wc -l
```

Quiz 3 answers

head file.txt

Kite rats kite cash REd kite

kite rats kite red caSh rats

```
grep '^[Kk]ite' file.txt | wc -l
```

The grep command does what?

Quiz 3 answers

head file.txt

Kite rats kite cash REd kite

kite rats kite red caSh rats

```
grep '^[Kk]ite' file.txt | wc -l
```

The grep command does what?

Counts K or k (upper/lower case) at the beginning of each line, which counts 2 in the file.

Quiz 3 answers

```
- echo "ATCCGCGCGCGCGGGCGCGCGCGTT" |  
grep -c "AT"
```

Does the command count all AT's in this sequence string?

T or F

Quiz 3 answers

```
- echo "ATCCGCGCGCGCGGGCGCGCGCGTT" |  
grep -c "AT"
```

Does the command count all AT's in this sequence string?

T or F

Quiz 3 answers

- `grep -c "AT" example.fasta` command is able to count all AT's in this line?

ATCCGCGCGCGCGGGCGCGCGCGCGCGGGCGCG
CGCGTT

T or F

Quiz 3 answers

- grep -c "AT" example.fasta command is able to count all AT's in this line?

ATCCGCGCGCGCGCGGCGCGCGCGCGCGCGCGGCGCG
CGCGTT

T or F

New question

- `grep -c "AT" example.fasta` command is able to count all AT's in this line?

ATCCGCGCGCGCGGGCGCGCGCGCGCGGGCGCG
CGCGTTAT

T or F

New question

- grep -c "AT" example.fasta command is able to count all AT's in this line?

ATCCGCGCGCGCGCGGCGCGCGCGCGCGCGGCGCG
CGCGTTAT

T or F

New question

- grep -c "AT" example.fasta command is able to count all AT's in this line?

ATCCGCGCGCGCGCGGCGCGCGCGCGCGCGGCGCG
CGCGTTAT

How would you count it?

New question

- `grep -c "AT" example.fasta` command is able to count all AT's in this line?

[illegible]

What this work?

grep -c "AT" example.fasta

New question

- `grep -c "AT" example.fasta` command is able to count all AT's in this line?

[illegible]

What this work?

```
grep -o -c "AT" example.fasta or  
grep -oc "AT" example.fasta
```


New question

- `grep -c "AT" example.fasta` command is able to count all AT's in this line?

[illegible]

What this work?

```
grep -Eo "AT" example.fasta --color
```

```
grep -o "AT" example.fasta
```

New question

- `grep -c "AT" example.fasta` command is able to count all AT's in this line?

[illegible]

What this work?

```
grep -Eo "AT" example.fasta | wc -l
```

```
grep -o "AT" example.fasta | wc -l
```

New question

- `grep -c "AT" example.fasta` command is able to count all AT's in this line?

[illegible]

What this work?

```
grep -o “^AT|AT$” example.fasta
```

New question

- grep -c "AT" example.fasta command is able to count all AT's in this line?

ATCCGCGCGCGCGGGC**GCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGCGTT****AT**

What this work?

```
grep -Eo “^AT|AT$” example.fasta | wc -l
```

```
egrep -o “^AT|AT$” example.fasta | wc -l
```

Bonus 3

- For the example.fasta file count both the number of AT and GC in one grep command
and in another command print the line number which they appear?

Command 1

```
grep -Eo "GC|AT" example.fasta | wc -l  
egrep -o "GC|AT" example.fasta | wc -l
```

Command 2

```
grep -nEo "GC|AT" example.fasta --color >>file.txt  
egrep -no "GC|AT" example.fasta --color >>file.txt
```

Bonus 3 extra questions

- For the example.fasta file count both the number of AT and GC in one grep command
and in another command print the line number which they appear?

How many GC and AT's are present count?

Number here:

command:

And, how many are GC and AT's individually?

Commands here:

Number here:

Bonus 3 extra questions

- For the example.fasta file count both the number of AT and GC in one grep command and in another command print the line number which they appear?

How many GC and AT's are present count?

Number here: 15

command: `egrep -o "GC|AT" example.fasta | wc -l`

And, how many are GC and AT's individually?

Commands here: `egrep -o "GC" example.fasta | wc -l`

`egrep -o "AT" example.fasta | wc -l`

Number here: GC = 8 and AT = 7

grep – syntax to hands of UNIX

grep [option] pattern file

Understanding Regular Expressions:

^ (Caret) match expression at the start of a line, as in ^A.

\$ (Dollar) match expression at the end of a line, as in A\$.

\ (Back Slash) turn off the special meaning of the next character, as in \^ . To look for a Caret “^” at the start of a line, the expression is ^\^.

[] (Brackets) match any one of the enclosed characters, as in [aeiou]. Use Hyphen “-” for a range, as in [0-9].

[^] match any one character except those enclosed in [], as in [^0-9].

. (Period) match a single character of any value, except end of line. So b.b will match “bob”, “bib”, “b-b”, etc.

* (Asterisk) match zero or more of the preceding character or expression. An asterisk matches zero or more of what precedes it. Thus [A-Z]* matches any number of upper-case letters, including none, while [A-Z][A-Z]* matches one or more upper-case letters.

grep examples

Write a grep regex command that counts 'kite' in beginning of the string only for kite_data.txt

grep examples

Write a grep regex command that counts 'kite' in beginning of the string only for kite_data.txt

```
grep '^kite' kite_data.txt | wc -l
```

or

```
grep -c '^kite' kite_data.txt
```

grep examples

Write a grep regex command that counts 'kite' in beginning of the string only for kite_data.txt

```
grep '^kite' kite_data.txt | wc -l
```

or

```
grep -c '^kite' kite_data.txt
```

What does the -w option do instead of -c?

grep examples

Write a grep regex command that counts 'kite or Kite' in beginning of the string only for kite_data.txt

grep examples

Write a grep regex command that counts 'kite or Kite' in beginning of the string only for kite_data.txt

```
grep '^[Kk]ite' kite_data.txt | wc -l
```

or

```
grep -c '^[Kk]ite' kite_data.txt
```

grep examples

Write a grep regex command that counts 'kite' in end of the string only for kite_data.txt

grep examples

Write a grep regex command that counts 'kite' in end of the string only for kite_data.txt

```
grep 'kite$' kite_data.txt | wc -l
```

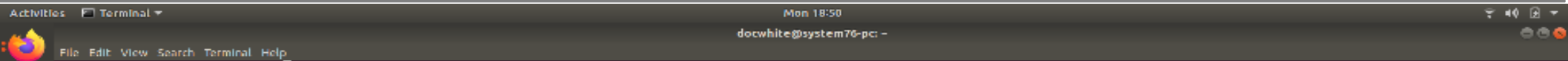
or

```
grep -c 'kite$' kite_data.txt
```

grep examples

Write a grep regex command that counts all words 'kite' but not 'Kite' for kite_data.txt

grep examples



File Edit View Search Terminal Help
(base) docwhite@system76-pc:~\$

Write a grep regex command that counts all words 'kite' but not 'Kite' for kite_data.txt

grep 'kite' kite_data.txt | wc -l

or

grep -c 'kite' kite_data.txt

grep examples

Write a grep regex command that counts all words 'kite' and 'Kite' for kite_data.txt

grep examples

Write a grep regex command that counts all words 'kite' and 'Kite' for kite_data.txt

```
grep '[Kk]ite' kite_data.txt | wc -l
```

or

```
grep -c '[Kk]ite' kite_data.txt
```

grep examples

Write a grep regex command that counts all words 'kite' and 'Kite' and 'red' for kite_data.txt

grep examples

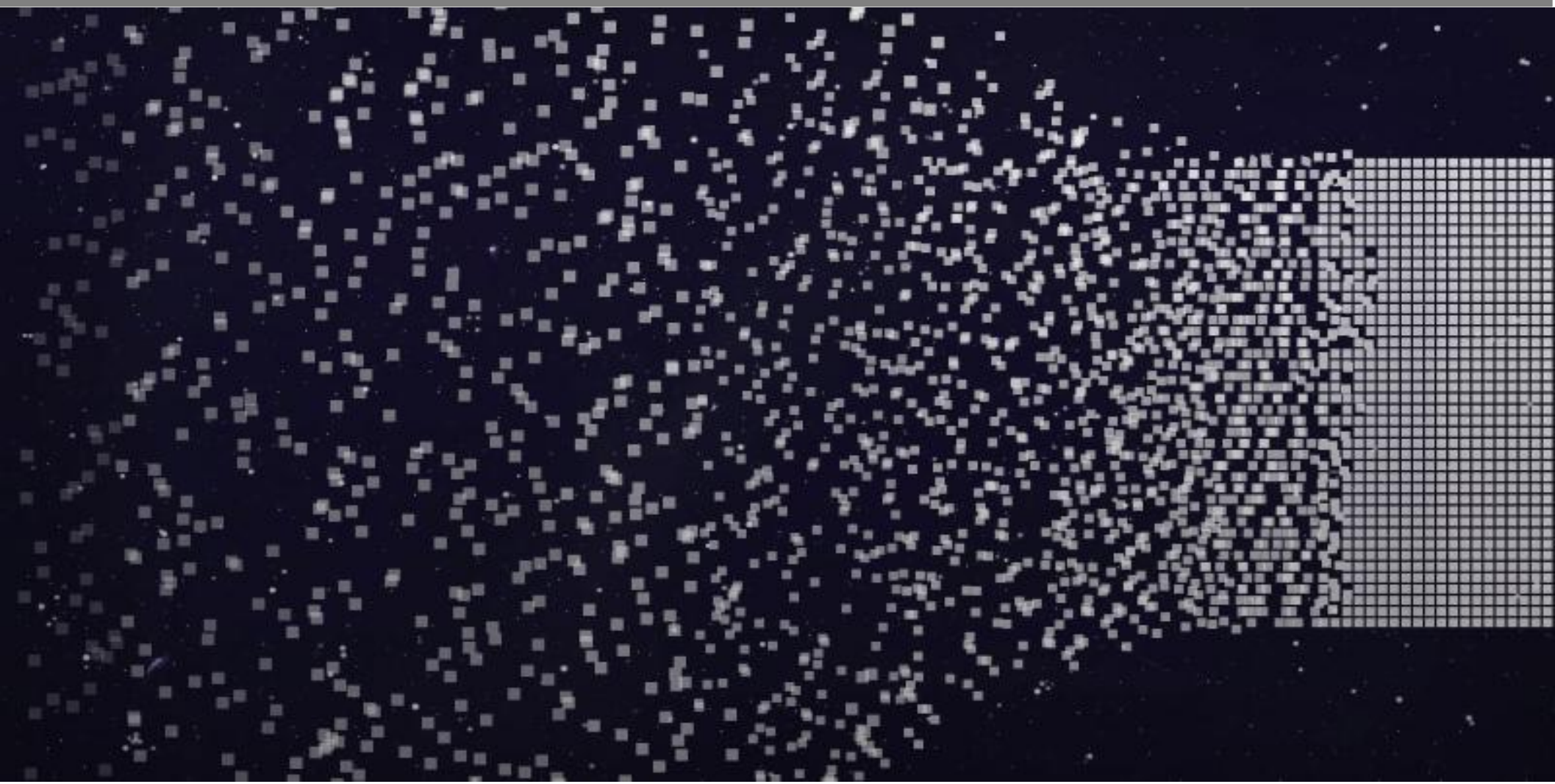
Write a grep regex command that counts all words 'kite' and 'Kite' and 'red' for kite_data.txt

```
grep -Eo '[Kk]ite|red' kite_data.txt | wc -l
```

or

```
egrep -o '[Kk]ite|red' kite_data.txt | wc -l
```

Data science

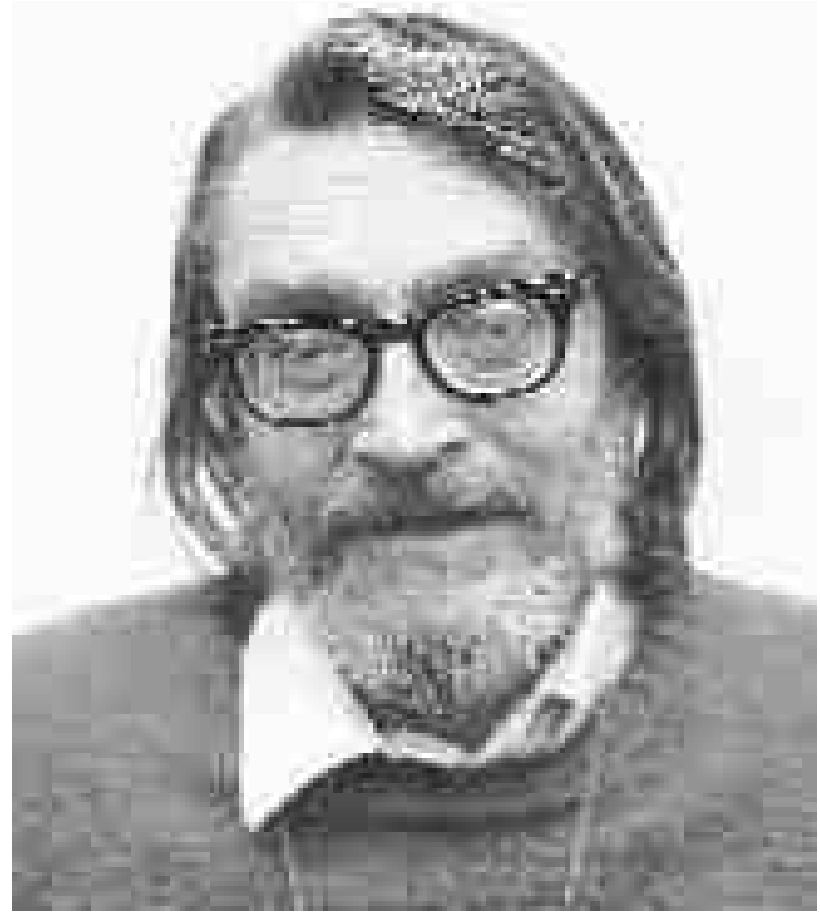


Data science

90% of data science is
DATA WRANGLING

Sed - Stream Editor

sed ("*stream editor*") is a Unix utility that parses and transforms text, using a simple, compact programming language. sed was developed from 1973-1974 by Lee E. McMahon of Bell Labs



Lee E.
McMahon

Sed - Stream Editor

sed ("*stream editor*") is sed was one of the earliest tools to support regular expressions, and remains in use for text processing, most notably with the substitution command. Written in C www.gnu.org/software/sed/



The screenshot shows the GNU Operating System website with a navigation bar containing links like 'ABOUT GNU', 'PHILOSOPHY', 'LICENSES', 'EDUCATION', 'SOFTWARE', 'DOCS', 'MALWARE', 'HELP GNU', and 'GNU ART'. The main heading is 'GNU sed'. Below it, a description states: 'sed (stream editor) is a non-interactive command-line text editor.' Two example commands are provided in a code block: 'Example: delete the 4th line in a file' with the command 's sed '4d' input.txt > output.txt', and 'Example: replace every occurrence of 'hello' with 'world' on lines 10 20' with the command 's sed '10,20s/hello/world/' input.txt > output.txt'. A paragraph explains that sed is commonly used to filter text, taking input, performing operations, and outputting modified text. A 'Getting Help' section lists resources: reading the manual, searching archives for questions and answers (including questions, development, and general discussion; mailing list; bug reports), and general sed resources available at SourceForge.net (including documentations, FAQ, one-liners, books, and example scripts). A sidebar on the right contains links for 'Getting Help', 'Downloads', 'Latest Source', 'Mailing Lists', and 'Maintainers'.

GNU Operating System
Supported by the Free Software Foundation

ABOUT GNU PHILOSOPHY LICENSES EDUCATION SOFTWARE DOCS MALWARE HELP GNU GNU ART

GNU sed

sed (stream editor) is a non-interactive command-line text editor.

```
# Example: delete the 4th line in a file
s sed '4d' input.txt > output.txt

# Example: replace every occurrence of 'hello' with 'world' on lines 10 20
s sed '10,20s/hello/world/' input.txt > output.txt
```

sed is commonly used to filter text, i.e., it takes text input, performs some operation (or set of operations) on it, and outputs the modified text. sed is typically used for extracting part of a file using pattern matching or substituting multiple occurrences of a string within a file.

Getting Help

- Read the manual locally using `info sed` or see the latest [online manual](#).
- Search the archives for previous questions and answers:
 - questions, development, and general discussion: [sed-devel mailing list](#);
 - bug reports: [bug-sed mailing list](#).
- General sed resources are available at [SourceForge.net](#), including:
 - [documentations](#), [FAQ](#), [one-liners](#),
 - [books](#), and
 - [example scripts](#).

- [Getting Help](#)
- [Downloads](#)
- [Latest Source](#)
- [Mailing Lists](#)
- [Maintainers](#)

Sed - Stream Editor

Popular alternative tools for plaintext string manipulation and "stream editing" include AWK and Perl.



The screenshot shows the GNU Operating System website with a navigation bar containing links like 'ABOUT GNU', 'PHILOSOPHY', 'LICENSES', 'EDUCATION', 'SOFTWARE', 'DOCS', 'MALWARE', 'HELP GNU', and 'GNU ART'. The main heading is 'GNU sed'. Below it, a paragraph describes sed as a non-interactive command-line text editor. A code block contains two examples: deleting the 4th line and replacing 'hello' with 'world' on lines 10-20. Another paragraph explains that sed is used to filter text. A 'Getting Help' section lists resources like the manual, archives, and SourceForge. A sidebar on the right contains links to 'Getting Help', 'Downloads', 'Latest Source', 'Mailing Lists', and 'Mainliners'.

GNU Operating System
Supported by the Free Software Foundation

ABOUT GNU PHILOSOPHY LICENSES EDUCATION SOFTWARE DOCS MALWARE HELP GNU GNU ART

GNU sed

sed (stream editor) is a non-interactive command-line text editor.

```
# Example: delete the 4th line in a file
$ sed '4d' input.txt > output.txt

# Example: replace every occurrence of 'hello' with 'world' on lines 10-20
$ sed '10,20s/hello/world/' input.txt > output.txt
```

sed is commonly used to filter text, i.e., it takes text input, performs some operation (or set of operations) on it, and outputs the modified text. sed is typically used for extracting part of a file using pattern matching or substituting multiple occurrences of a string within a file.

Getting Help

- Read the manual locally using `info sed` or see the latest [online manual](#).
- Search the archives for previous questions and answers:
 - questions, development, and general discussion: [sed-devel mailing list](#);
 - bug reports: [bug-sed mailing list](#).
- General sed resources are available at [SourceForge.net](#), including:
 - [documentations](#), [FAQ](#), [one-liners](#),
 - [books](#), and
 - [example scripts](#).

- [Getting Help](#)
- [Downloads](#)
- [Latest Source](#)
- [Mailing Lists](#)
- [Mainliners](#)

Sed - Stream Editor

```
s/he/she/g
```

Sed - Stream Editor

sed OPTIONS... [SCRIPT] [INPUTFILE...]

- version (Print out the version of sed)
- help (help page)
- n, --quiet, --silent (suppress automatic printing of pattern space)
- debug
- e script, --expression=script (Add the commands in script to the set of commands to be run)
- i [SUFFIX], --in-place[=SUFFIX] (This option specifies that files are to be edited in-place)

Sed - Stream Editor

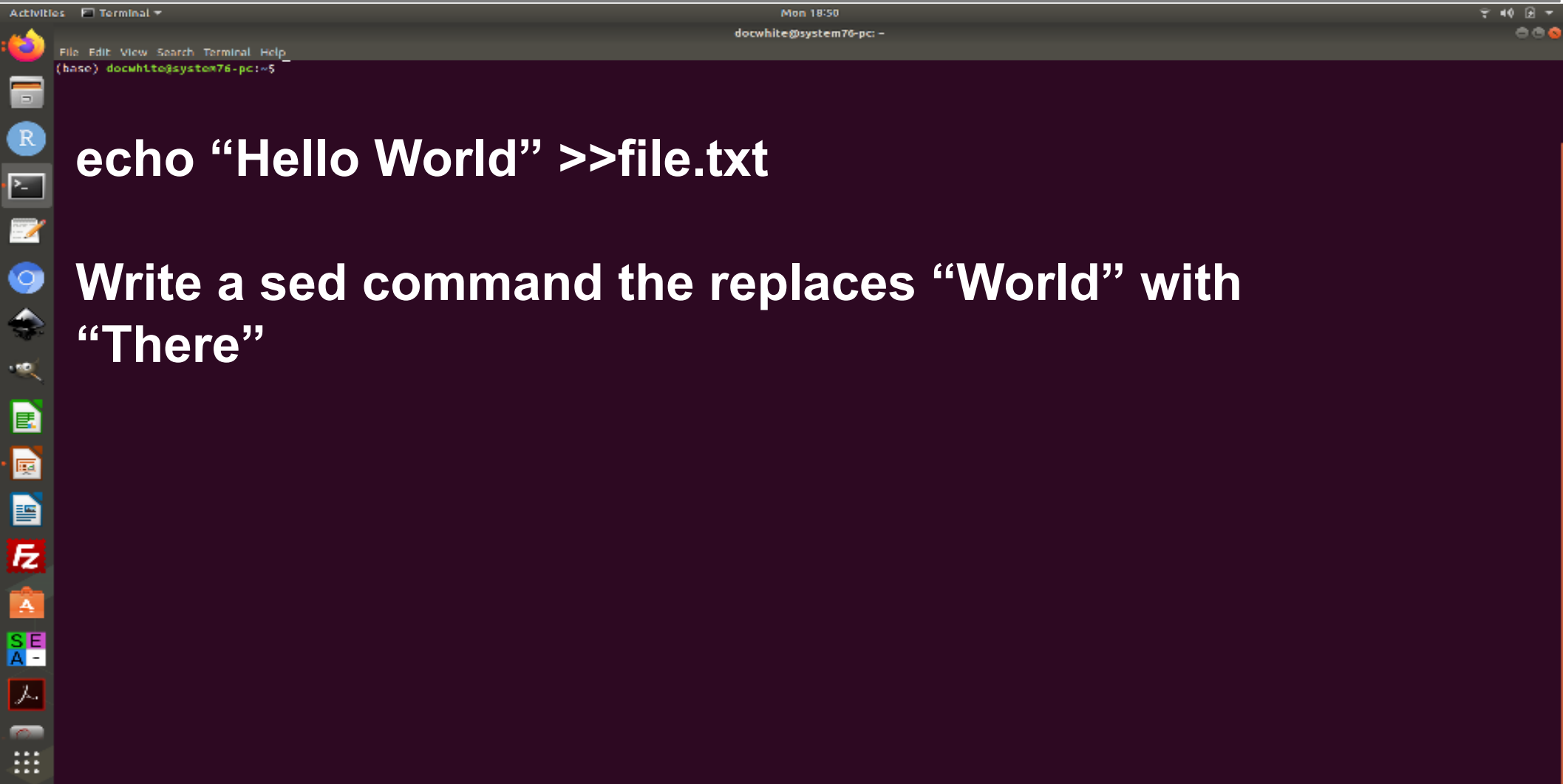
Substitution command

S command swiss army knife

```
sed 's/regexp/replacement/g' inputFileName >  
outputFileName
```

The **s** stands for substitute, while the **g** stands for global, which means that all matching occurrences in the line would be replaced.

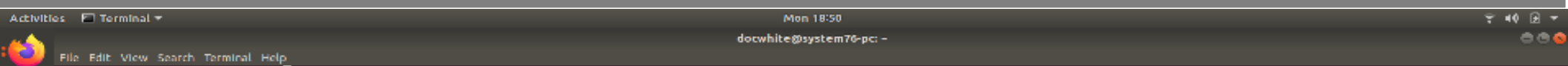
sed examples



```
echo "Hello World" >>file.txt
```

Write a sed command the replaces “World” with
“There”

sed examples



echo “Hello World” >>file.txt

**Write a sed command the replaces “World” with
“There”**

sed ‘s/World/There/g’ file.txt >>file1.txt

What happens when I do

sed ‘s/World/There/g’ file.txt

sed examples

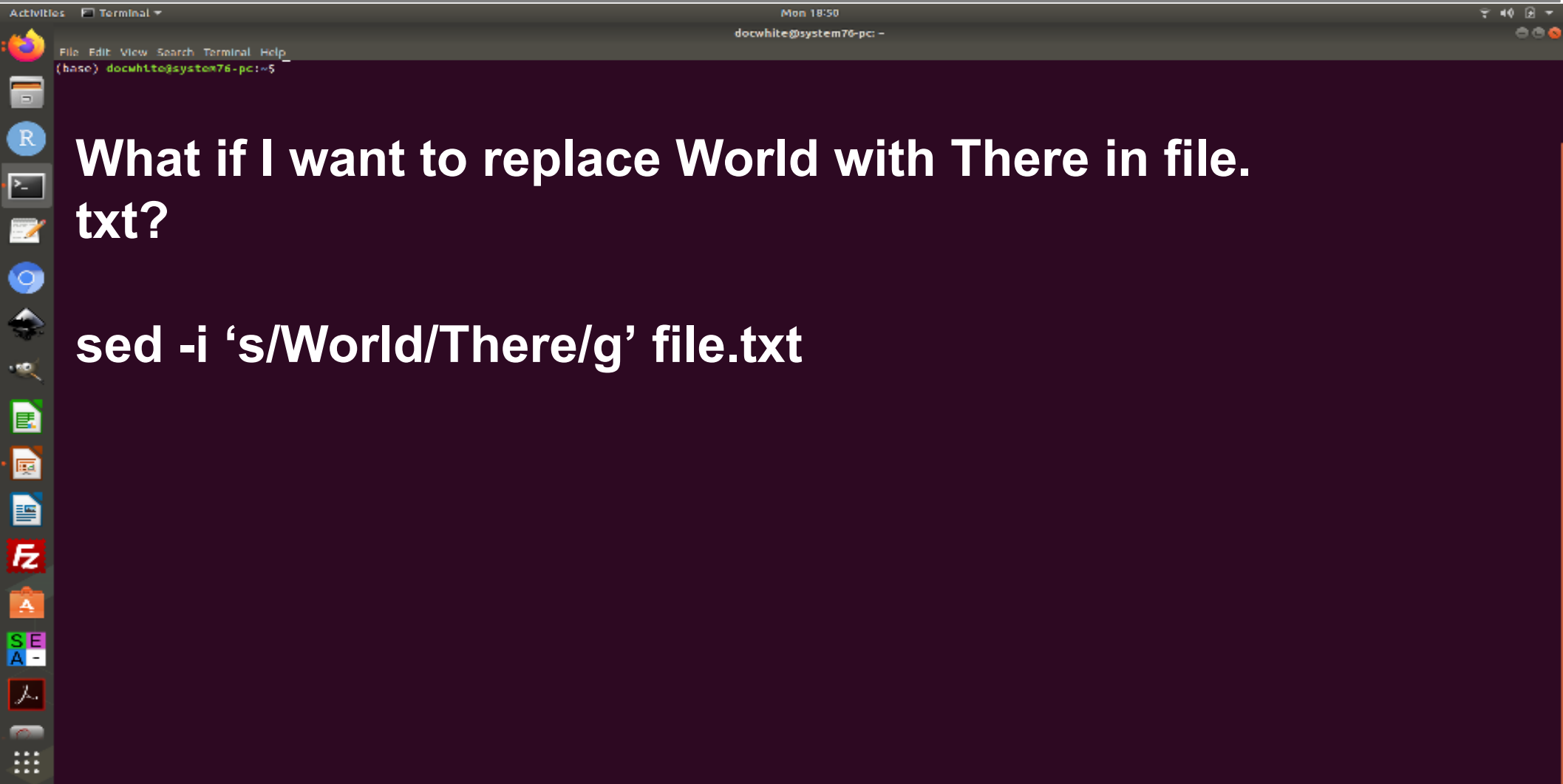
What happens when I do ?

sed 's/World/There/g' file.txt

Then

More file.txt?

sed examples



What if I want to replace World with There in file.txt?

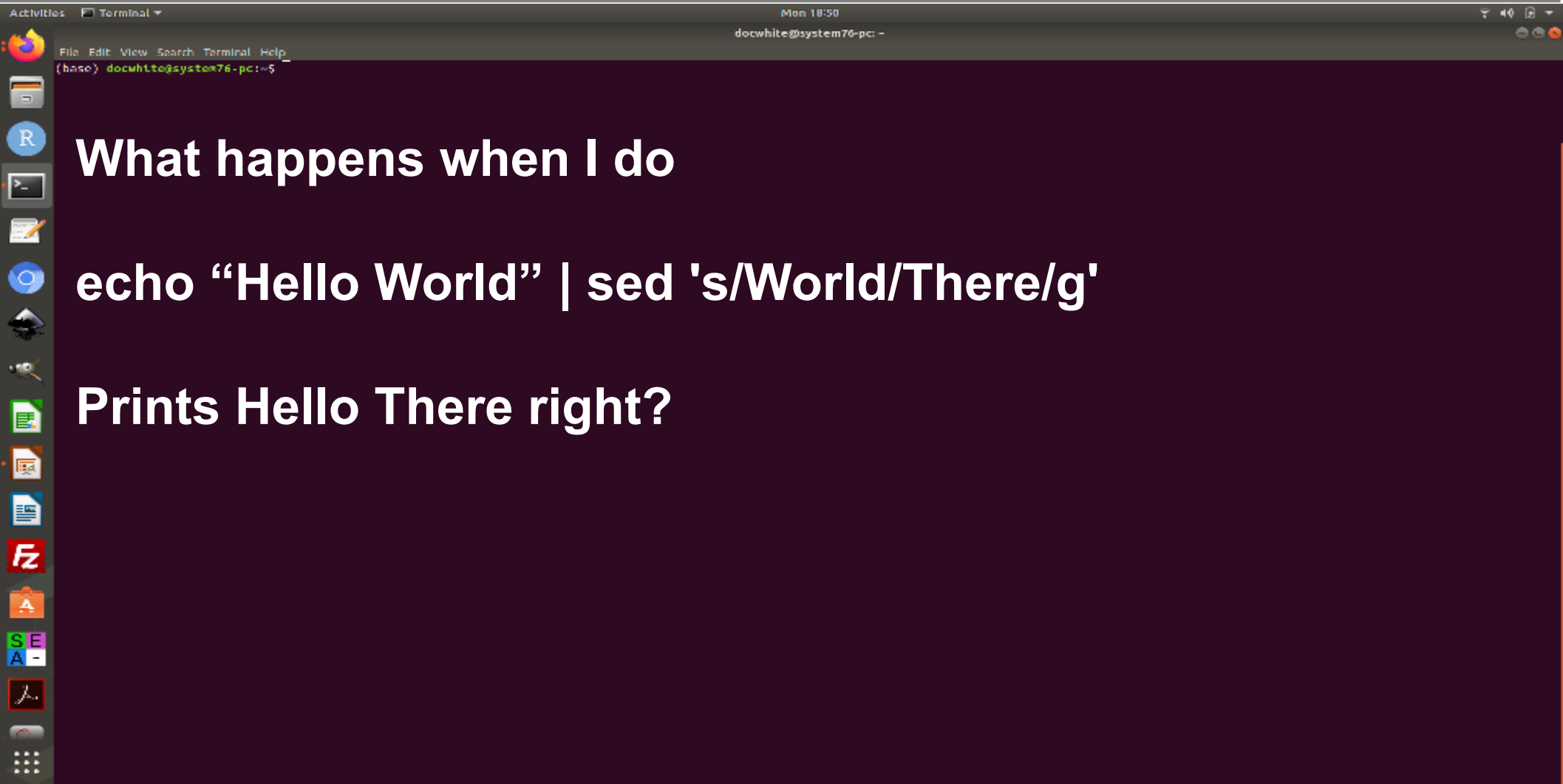
sed -i 's/World/There/g' file.txt

sed examples

What happens when I do

echo "Hello World" | sed 's/World/There/g'

sed examples



What happens when I do

echo "Hello World" | sed 's/World/There/g'

Prints Hello There right?

Sed – S swiss army knife

sed 's/regexp/replacement/**flags**'.

\L - Turn the replacement to lowercase until a \U or \E is found

\l - Turn the next character to lowercase,

\U - Turn the replacement to uppercase until a \L or \E is found,

\u - Turn the next character to uppercase,

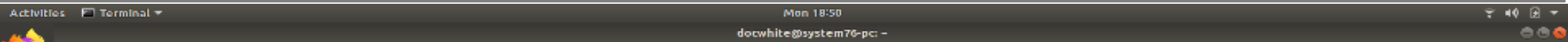
\E - Stop case conversion started by \L or \U.

g - Apply the replacement to all matches to the regexp, not just the first.

d - Delete the pattern space; immediately start next cycle.

a comment, until the next newline.

sed examples



File Edit View Search Terminal Help
(base) docwhite@system76-pc:~\$

From the kite_data.txt

Write a sed command that makes all the lower
case upper case?

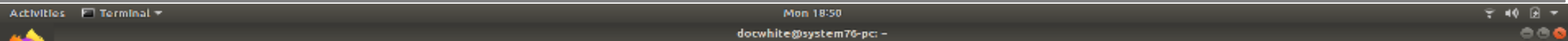
sed examples

From the kite_data.txt

Write a sed command that makes all the lower case upper case?

`sed 's/[a-z]/\U&/g'`

sed examples



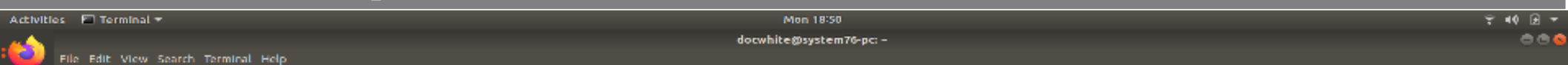
File Edit View Search Terminal Help
(base) docwhite@system76-pc:~\$

From the kite_data.txt

Write a sed command that makes all the upper
case lower case?

sed 's/[A-Z]/\u0026;/g'

sed examples



File Edit View Search Terminal Help
(base) docwhite@system76-pc:~\$

From the kite_data.txt

Write a sed command that makes all the upper case lower case?

sed 's/[A-Z]/l&/g'

or

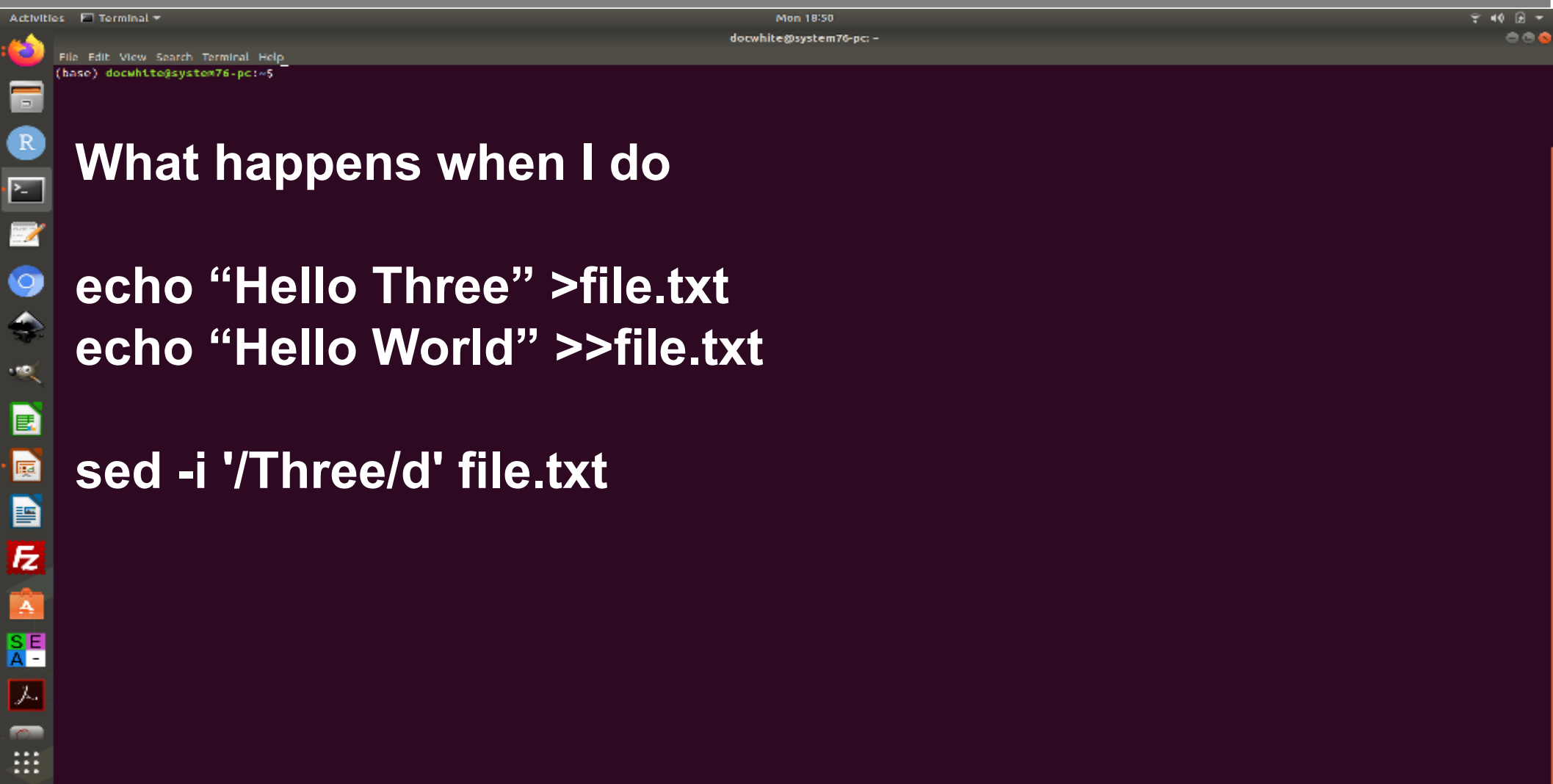
sed 's/[A-Z]/L&/g'

Sed – delete

sed **'/^ */d'** inputFileName.

- The caret (^) matches the beginning of the line.
- The dollar sign (\$) matches the end of the line.
- The asterisk (*) matches zero or more occurrences of the previous character.
- The plus (+) matches one or more occurrence(s) of the previous character.
- The question mark (?) matches zero or one occurrence of the previous character.
- The dot (.) matches exactly one character.

sed examples



What happens when I do

```
echo "Hello Three" >file.txt
```

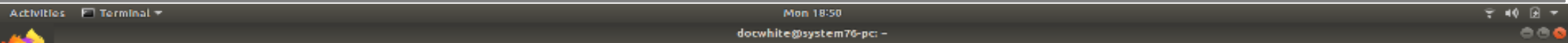
```
echo "Hello World" >>file.txt
```

```
sed -i '/Three/d' file.txt
```

sed examples

I want you do delete all the empty lines using sed?

sed examples



File Edit View Search Terminal Help
(base) docwhite@system76-pc:~\$

I want you to delete all the empty lines using
sed?

sed '/^\$/d' empty_lines.txt

Bonus 4

- Delete all the empty lines in the empty lines file with
 - grep
 - awk
- Delete all the 'all white space' with grep
 - grep
 - awk

Also, in python (think Pandas)

Quiz 4

- On canvas now