BINF2111 – Introduction to Bioinformatics Computing

BASH 101 – Loops part duex

Richard Allen White III, PhD
RAW Lab

Lecture 10 – Tuesday Sep 21st, 2023

Learning Objectives

- Review quiz/bonus (SAM files)

- File conversions (awk/perl)

- Specific lines UNIX/BASH

- Review Bash for loops (C-style/Regular)

- Quiz 10

Quiz 9 answers

Write a one-liner that counts the number of times Steven is left of Jose?

```
more file.csv
Steven,Jose
Steven,Jose
Steven,Jose
```

```
awk 'Jose*Steven' file.csv | wc -l
awk '/jose\tSteven' file.csv | wc -l
egrep -o 'Steven.Jose' file.csv | wc -l
egrep -o 'Steven\tJose' file.csv | wc -l
```

Quiz 9 answers

Write a one-liner that counts the number of times Steven is left of Jose?

```
more file.csv
Steven,Jose
Steven,Jose
Steven,Jose
```

```
awk 'Jose*Steven' file.csv | wc -l
awk '/jose\tSteven' file.csv | wc -l
egrep -o 'Steven.Jose' file.csv | wc -l
egrep -o 'Steven\tJose' file.csv | wc -l
```

How would we do this in AWK?

Quiz 9 answers

Write a one-liner that counts the number of times Steven is left of Jose?

```
more file.csv
Steven,Jose
Steven,Jose
Steven,Jose
```

How would we do this in AWK?

```
awk -F ',' 'BEGIN { count=0 } $1 == "Steven" && $2 == "Jose"
{ count++ } END { print count }' file.csv
```

Quiz 9 answers

Which command doesn't convert tsv to csv?

Command 1

```
sed 's/\t/,/g' file.tsv >file.csv
```

Command 2

```
cat file.tsv | tr -s '\t' ',' >file.csv
```

Command 3

```
awk -F '\t' -vOFS=’,’ '{$1=$1}1' file.tsv >file.csv
```

Command 4

```
awk -F '\t' -vOFS=’,’ '{$1= $1}1' file.tsv >file.csv
```

All convert tsv to csv

Quiz 9 answers

Which command doesn't convert tsv to csv?

Command 1

```
sed 's/\t/,/g' file.tsv >file.csv
```

Command 2

```
cat file.tsv | tr -s '\t' ',' >file.csv
```

Command 3

```
awk -F '\t' -vOFS=’,’ '{$1=$1}1' file.tsv >file.csv
```

Command 4

```
awk -F '\t' -vOFS=’,’ '{$1= $1}1' file.tsv >file.csv
```

All convert tsv to csv

The file conversation problem

Provide unique commands to convert a tsv to csv?

```
awk '{gsub("\t", ","); print}' file.tsv
```

Try the name_game.tsv and tsv file.

Also, it works similar to sed and similar syntax to tr (translate):

```
awk '{gsub(“original”, "replace"); print}' file.txt
```

OR

```
awk '{gsub(/original/, "replace"); print}' file.txt
```

Another way?

The file conversation problem

Provide unique commands to convert a tsv to csv?

```
perl -pi -e 's/\t/,/g' name_game.tsv (replaces original -i)
perl -p -e 's/\t/,/g' name_game.tsv (doesn’t replace original)
```

Try the name_game.tsv and csv file.

Also, it works just like sed try this!

```
sed 's/\t/,/' name_game.tsv
```

OR

```
perl -p -e 's/\t/,/' name_game.tsv
```

Quiz 9

- Write a one-liner that counts the number of times Steven is left of Jose?

```
more file.tsv

Steven  Jose
Steven  Jose
Steven  Jose
Steven  Jose
```

Quiz 9

- Write a one-liner that counts the number of times Steven is left of Jose?

more file.**tsv**

```
Steven  Jose
Steven  Jose
Steven  Jose
Steven  Jose
```

What is a tsv?

Quiz 9

- Write a one-liner that counts the number of times Steven is left of Jose?

```
more file.tsv

Steven	Jose
Steven	Jose
Steven	Jose
Steven	Jose
```

What is a tsv? Tab separated file
Tabs are different from spaces

Quiz 9

- Write a one-liner that counts the number of times Steven is left of Jose?

```
more file.tsv

Steven	Jose
Steven	Jose
Steven	Jose
Steven	Jose
```

```
egrep -o 'Steven.Jose' file.tsv | wc -l
```

Quiz 9

- Write a one-liner that counts the number of times Steven is left of Jose?

more file.**txt**

```
Steven  Jose
Steven  Jose
Steven  Jose
Steven  Jose
```

```
egrep -o 'Steven. Jose' file.tsv | wc -l
```

Quiz 9

- Write a one-liner that counts the number of times Steven is left of Jose?

```
more file.txt

Steven  Jose
Steven  Jose
Steven  Jose
Steven  Jose
```

How do you check? With sed.

Quiz 9

- Write a one-liner that counts the number of times Steven is left of Jose?

```
more file.txt

Steven  Jose
Steven  Jose
Steven  Jose
Steven  Jose
```

Sed ‘s/ /,/g’ file.txt | more (converts to csv)

Quiz 9

- Write a one-liner that counts the number of times Steven is left of Jose?

```
more file.txt

Steven  Jose
Steven  Jose
Steven  Jose
Steven  Jose
```

sed ‘s/\t/,/g’ file.txt | more (converts to csv)

Quiz 9

I have this file:

```
>chr1_geneA
ATGCTAAGGCTATCTTGACAACTGACTGCCTAG
>chr1_geneB
ATGCTAAGGCTATGTTGGCAACTGACTCCCTAG
```

How do I extract this sequence with grep?

Quiz 9

I have this file:

```
>chr1_geneA
ATGCTAAGGCTATCTTGACAACTGACTGCCTAG
>chr1_geneB
ATGCTAAGGCTATGTTGGCAACTGACTCCCTAG
```

How do I extract this sequence with grep?

grep 'ATG.*ATG' file.fna

Another way?

Quiz 9

I have this file:

```
>chr1_geneA
ATGCTAAGGCTATCTTGACAACTGACTGCCTAG
>chr1_geneB
ATGCTAAGGCTATGTTGGCAACTGACTCCCTAG
>chr1_geneC
ATGCTAAGGCTATGTTGGCAACTGACTCCCTAGATG
```

How do I extract this sequence with grep?

```
grep '\(.*ATG\)\{3\}' test.fna | more
```

Bonus 8

- Use grep to convert sam file to a fastq file?

Bonus 8

- Use grep to convert sam file to a fastq file?

What is a SAM file?

The **Sequence Alignment/Map (SAM)** is a file format to save alignment information of short reads mapped against reference sequences. It usually starts with a header section followed by alignment information as tab separated lines for each read.

Bonus 8

- Use grep to convert sam file to a fastq file?

Header section

```
@HD    VN:1.3    SO:coordinate
```

Tab-delimited read alignment information lines

```
readID43GYAX15:7:1:1202:19894/1    256    contig43    613960    1    65M    *    0    0
CCAGCGCGAACGAAATCCGCATGCGTCTGGTCGTTGCACGGAACGGCGGCGGTGTGATGC
ACGGC    EDDEEDEE=EE?DE??DDDBADEBEFFFDBEFFEBCBC=?BEEEE@=:?::?7?:8-6?7?
@??#    AS:i:0    XS:i:0  XN:i:0  XM:i:0  XO:i:0  XG:i:0  NM:i:0  MD:Z:65  YT:Z:UU
```

Bonus 8

- Use grep to convert sam file to a fastq file?

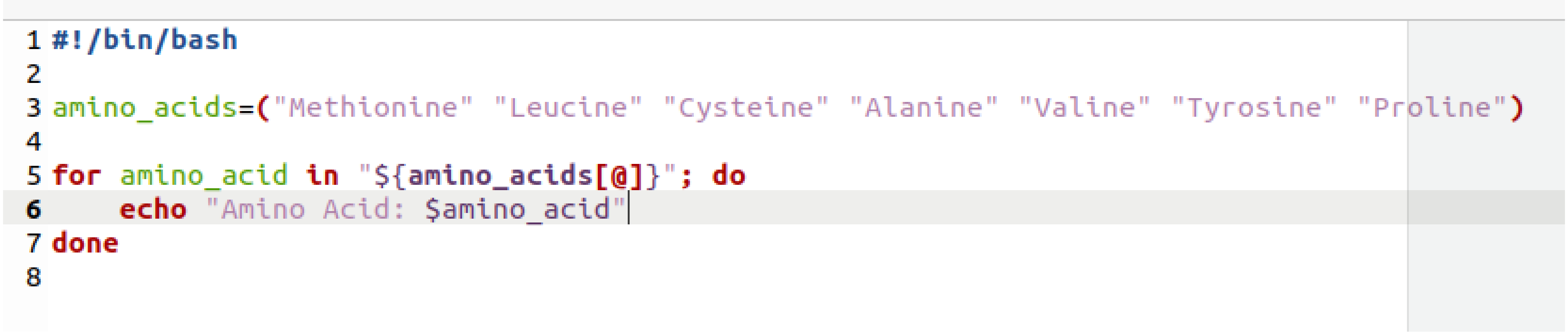
```
cat samplename.nomapping.sam | grep -v ^@ | awk '{print "@"$1"\n"$10"\n+\n"$11}' > unmapped/samplename.fastq
```

Lab bonus 1

Write a BASH code to iterate through the amino acid array.
Methionine Leucine Cysteine Alanine Valine Tyrosine Proline

Lab bonus 1

Write a BASH code to iterate through the amino acid array.
Methionine Leucine Cysteine Alanine Valine Tyrosine Proline



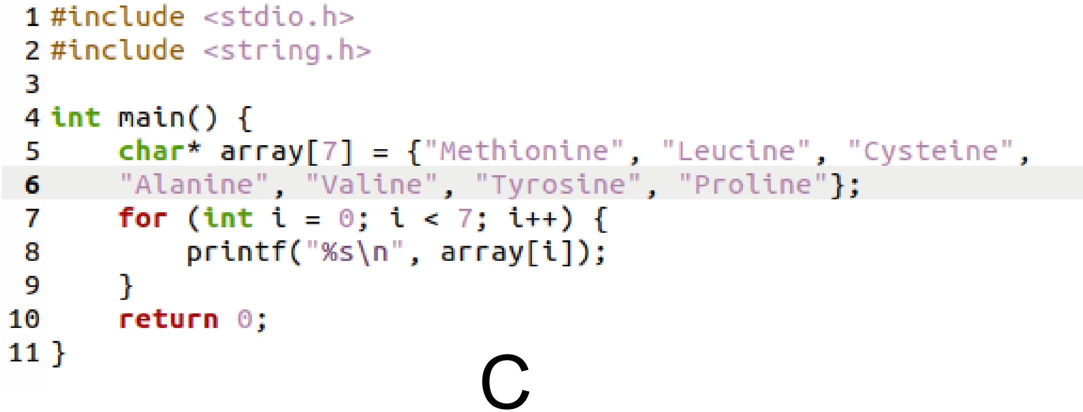
```bash
#!/bin/bash

amino_acids=("Methionine" "Leucine" "Cysteine" "Alanine" "Valine" "Tyrosine" "Proline")

for amino_acid in "${amino_acids[@]}"; do
    echo "Amino Acid: $amino_acid"
done

```

Lab bonus 2 – C and C++

Write a BASH code to iterate through the amino acid array. Methionine Leucine Cysteine Alanine Valine Tyrosine Proline in another language?

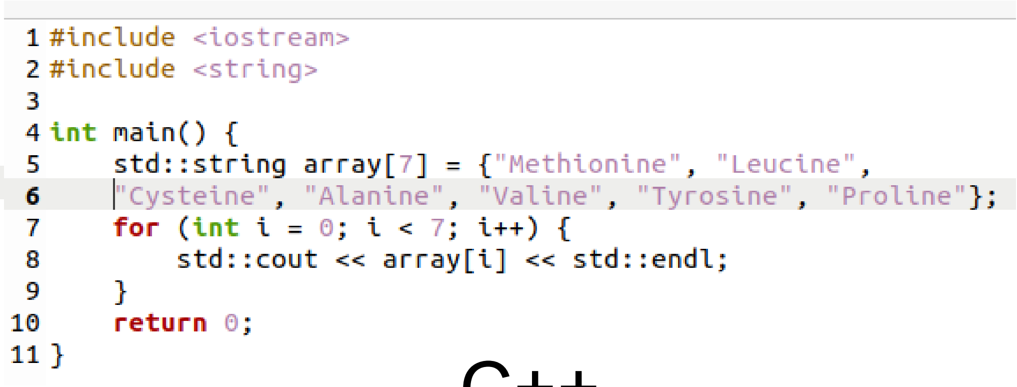
Lab bonus 2 – C and C++

Write a BASH code to iterate through the amino acid array. Methionine Leucine Cysteine Alanine Valine Tyrosine Proline in another language?



```c
#include <stdio.h>
#include <string.h>

int main() {
    char* array[7] = {"Methionine", "Leucine", "Cysteine",
    "Alanine", "Valine", "Tyrosine", "Proline"};
    for (int i = 0; i < 7; i++) {
        printf("%s\n", array[i]);
    }
    return 0;
}
```

C



```cpp
#include <iostream>
#include <string>

int main() {
    std::string array[7] = {"Methionine", "Leucine",
    "Cysteine", "Alanine", "Valine", "Tyrosine", "Proline"};
    for (int i = 0; i < 7; i++) {
        std::cout << array[i] << std::endl;
    }
    return 0;
}
```

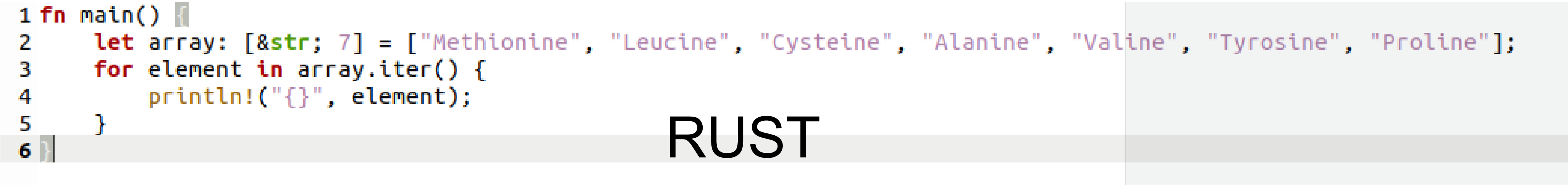
C++

gcc bonusar2.c -o bonusar2c
./bonussar2c

g++ -o bonusar2cp bonusar2.cpp
./bonusar2cp

Lab bonus 2 - Rust

Write a BASH code to iterate through the amino acid array.
Methionine Leucine Cysteine Alanine Valine Tyrosine Proline
in another language?



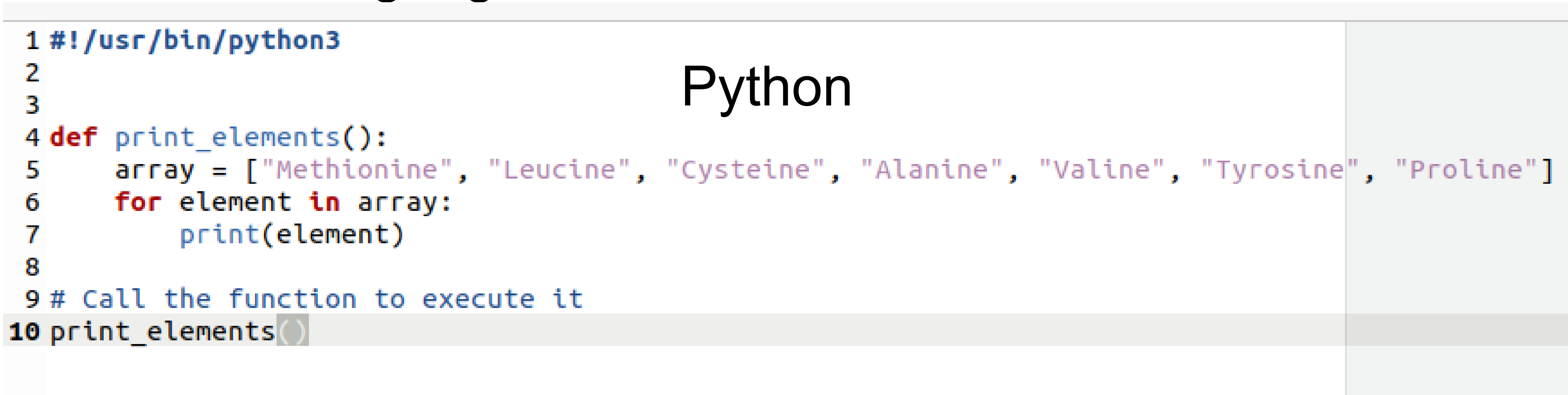
```
fn main() {
    let array: [&str; 7] = ["Methionine", "Leucine", "Cysteine", "Alanine", "Valine", "Tyrosine", "Proline"];
    for element in array.iter() {
        println!("{}", element);
    }
}
```

RUST

```
rustc bonus2rust.rs
./bonus2rust
```

Lab bonus 2 - Python

Write a BASH code to iterate through the amino acid array.
Methionine Leucine Cysteine Alanine Valine Tyrosine Proline
in another language?



```
#!/usr/bin/python3


def print_elements():
    array = ["Methionine", "Leucine", "Cysteine", "Alanine", "Valine", "Tyrosine", "Proline"]
    for element in array:
        print(element)

# Call the function to execute it
print_elements()
```

Python

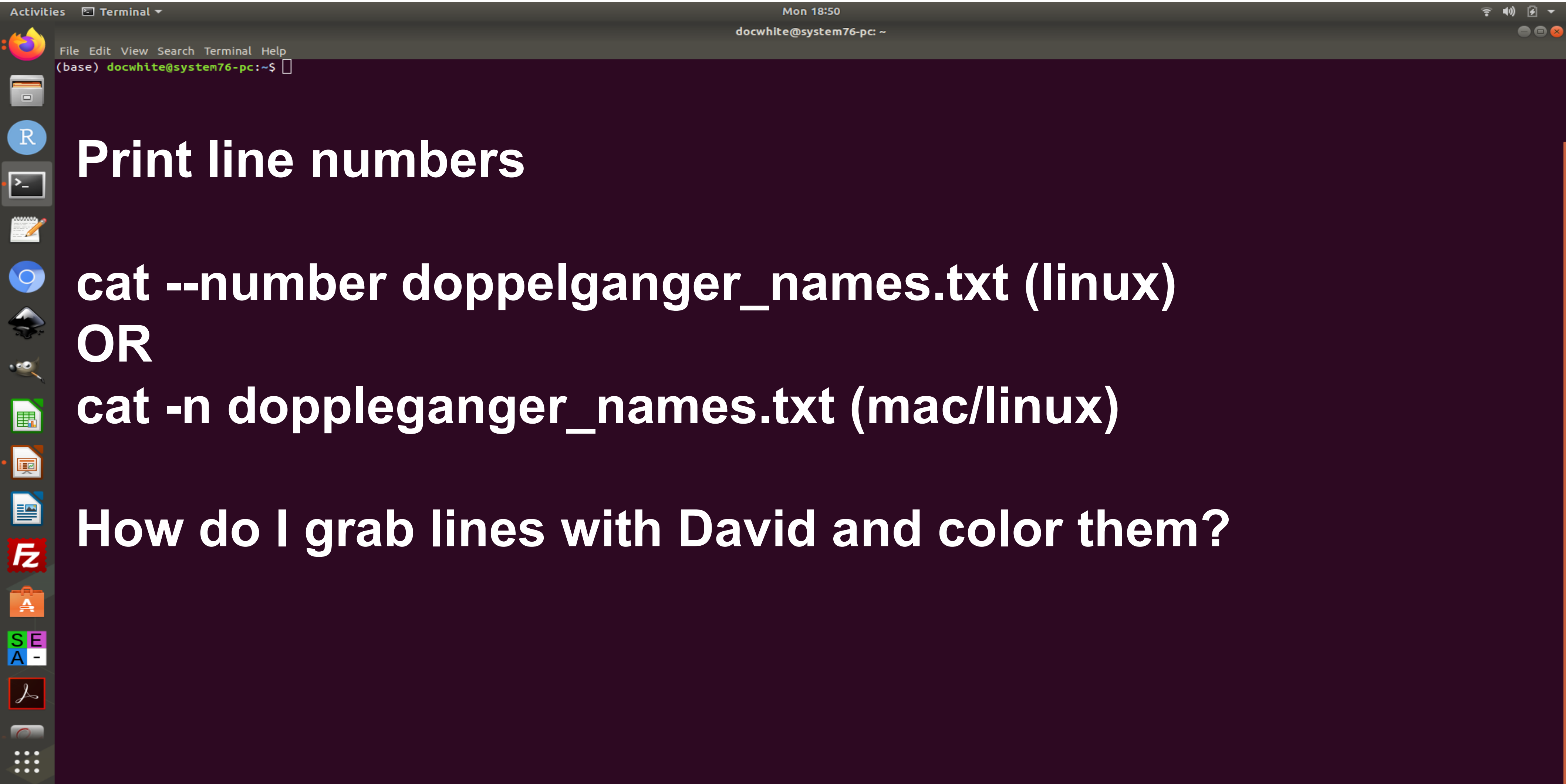
Specific lines in UNIX/BASH

- How do I print line numbers?

- What if you need specific lines in a file?

- What how do you do a range of specific lines?

Specific lines



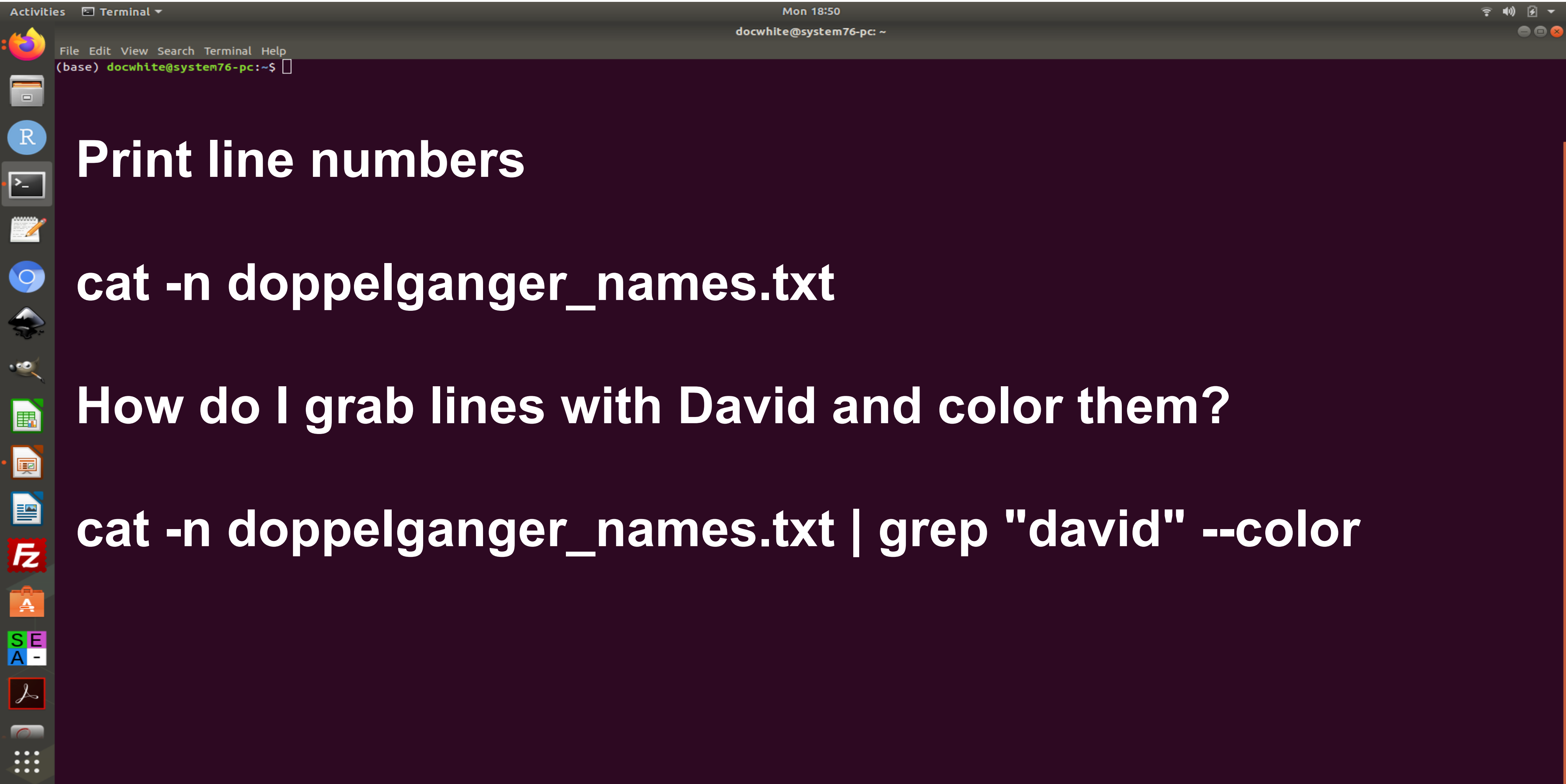
```
(base) docwhite@system76-pc:~$ 
```

Print line numbers

cat --number doppelganger_names.txt (linux)
OR
cat -n doppleganger_names.txt (mac/linux)

How do I grab lines with David and color them?

Specific lines



```
(base) docwhite@system76-pc:~$
```

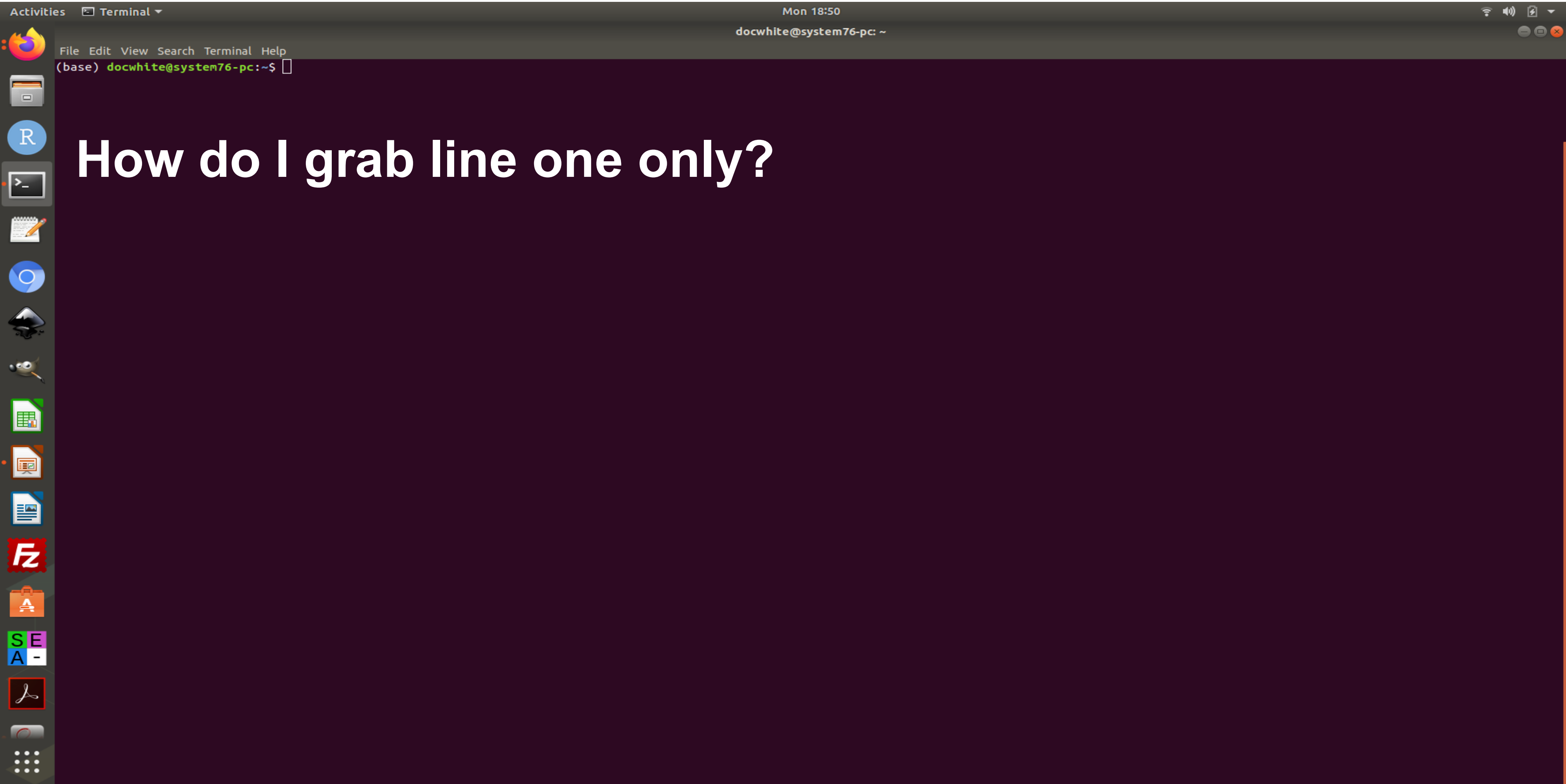
Print line numbers

```
cat -n doppelganger_names.txt
```

How do I grab lines with David and color them?

```
cat -n doppelganger_names.txt | grep "david" --color
```

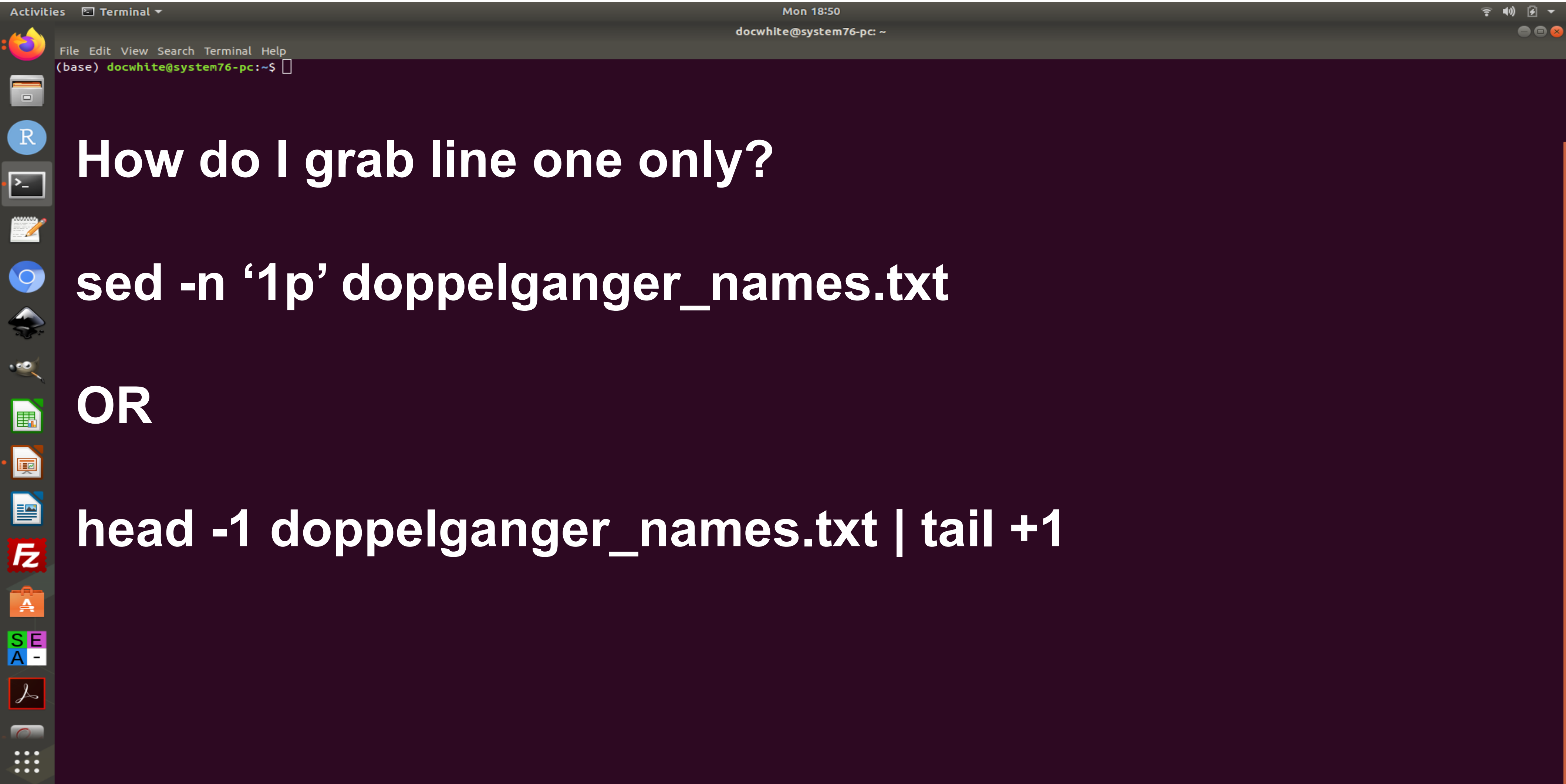
Specific lines



```
(base) docwhite@system76-pc:~$ 
```

How do I grab line one only?

Specific lines



```
(base) docwhite@system76-pc:~$
```

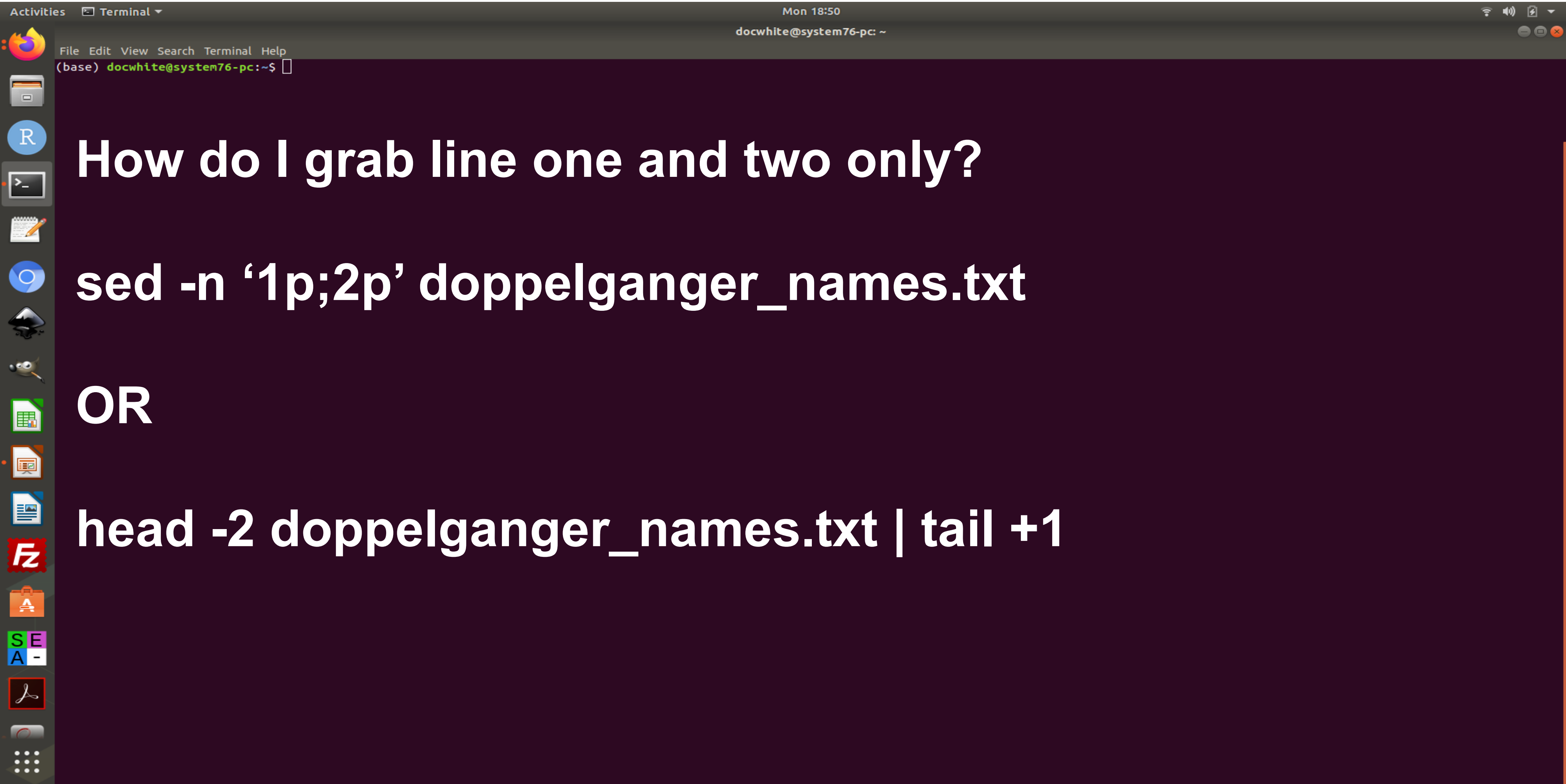
How do I grab line one only?

```
sed -n ‘1p’ doppelganger_names.txt
```

OR

```
head -1 doppelganger_names.txt | tail +1
```

Specific lines



```
(base) docwhite@system76-pc:~$ 
```

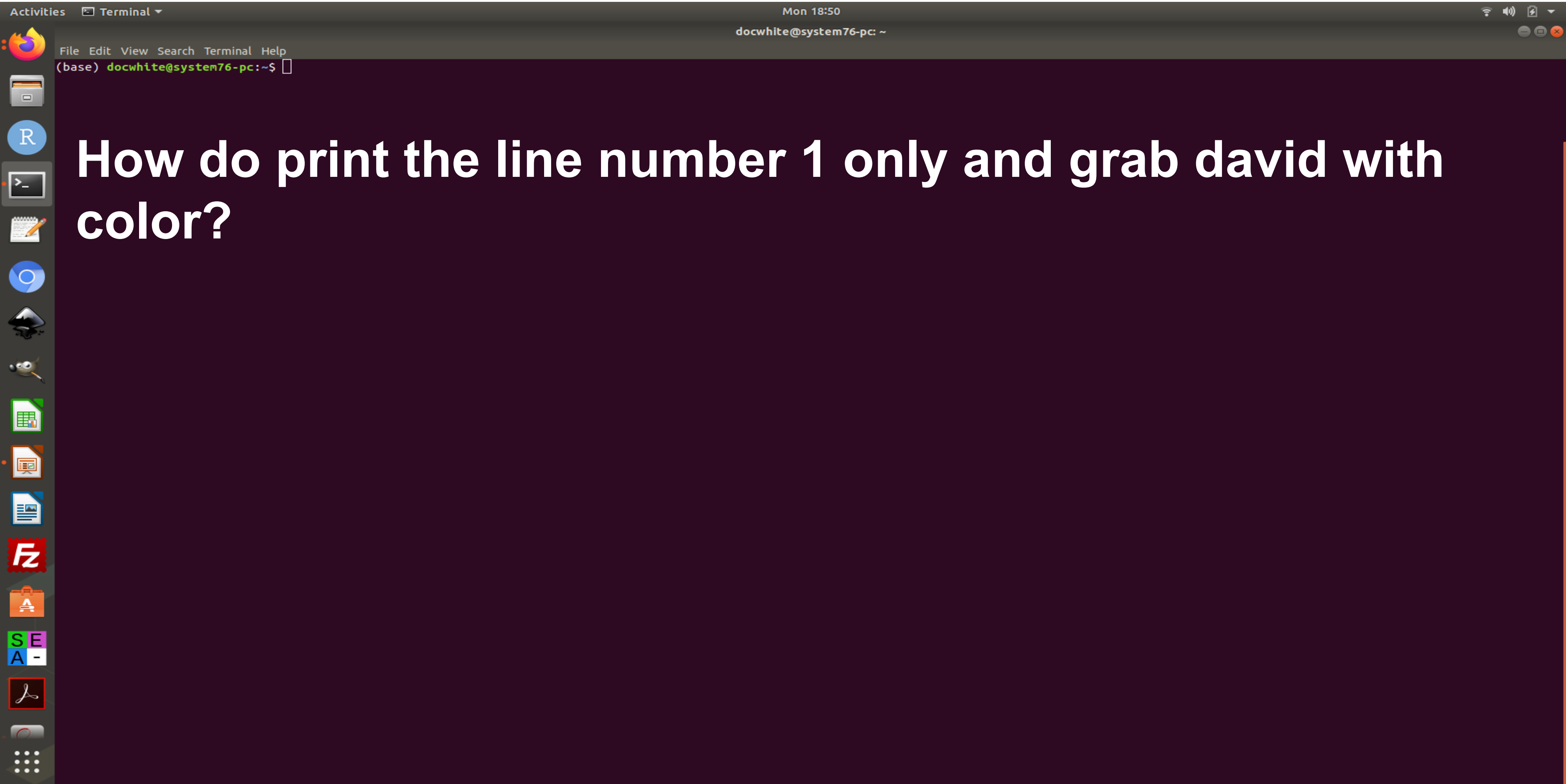
How do I grab line one and two only?

```
sed -n ‘1p;2p’ doppelganger_names.txt
```

OR

```
head -2 doppelganger_names.txt | tail +1
```

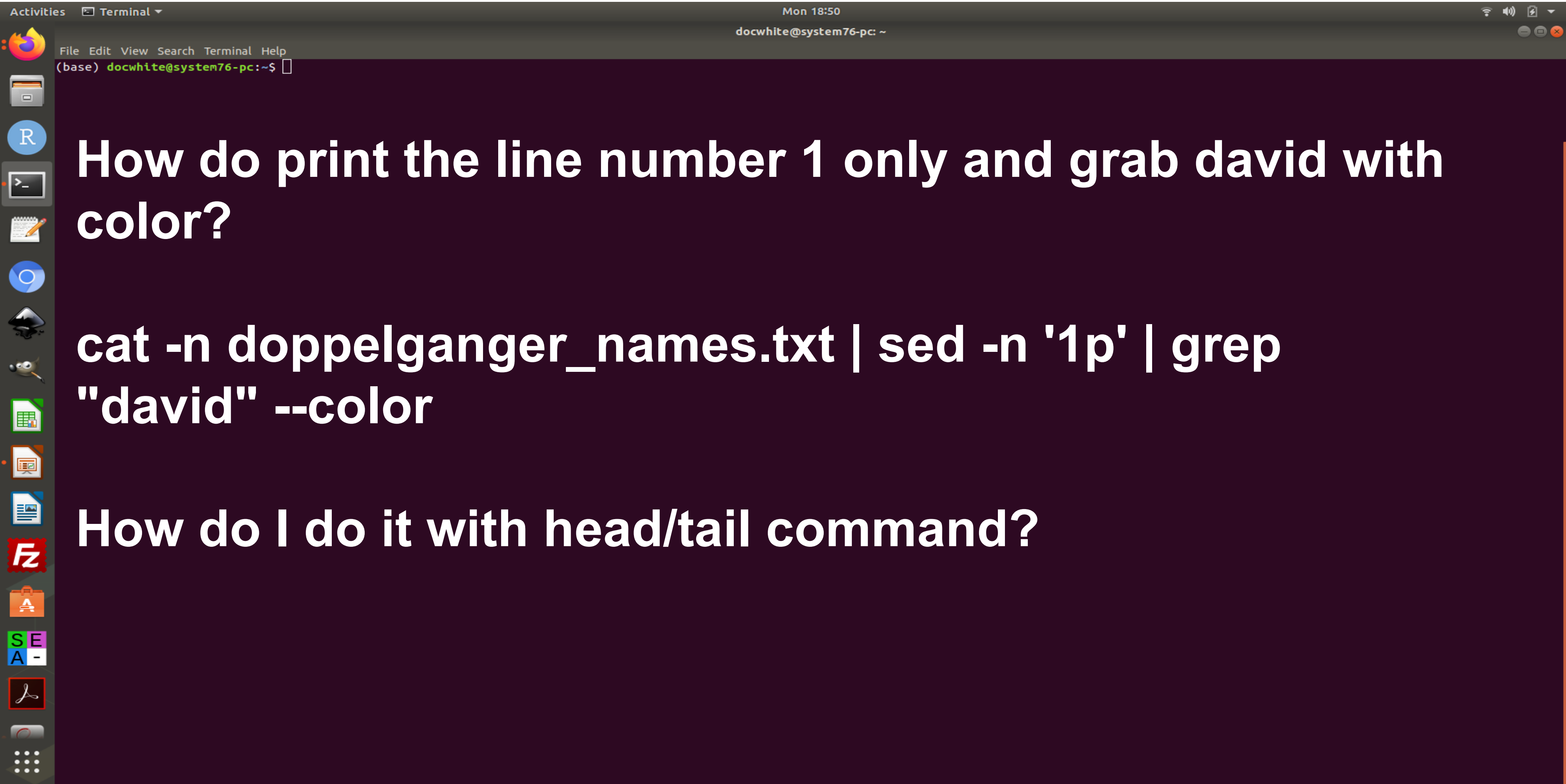
Specific lines



```
(base) docwhite@system76-pc:~$ 
```

How do print the line number 1 only and grab david with color?

Specific lines



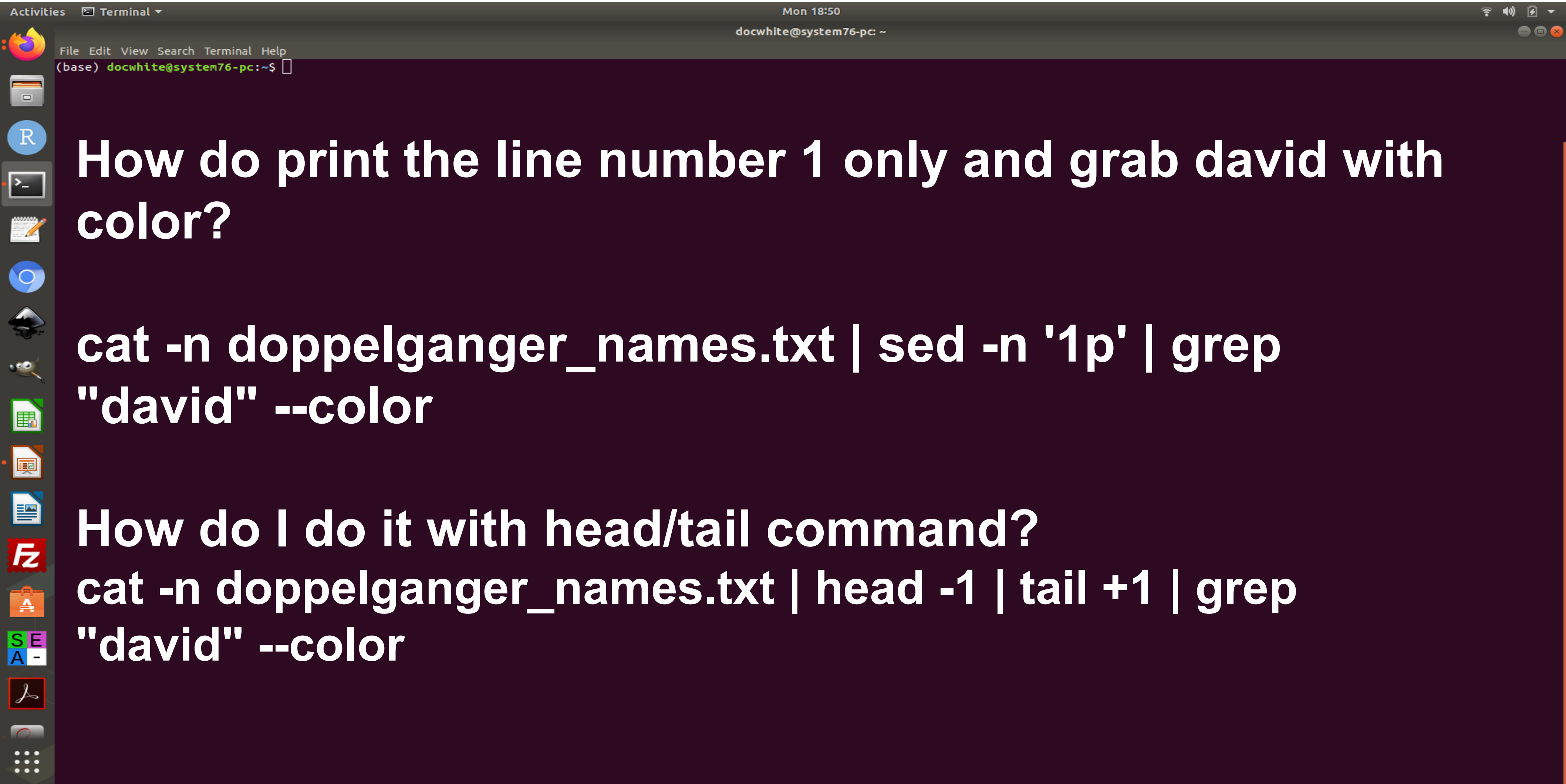
```
(base) docwhite@system76-pc:~$
```

How do print the line number 1 only and grab david with color?

```
cat -n doppelganger_names.txt | sed -n '1p' | grep "david" --color
```

How do I do it with head/tail command?

Specific lines



```
(base) docwhite@system76-pc:~$
```

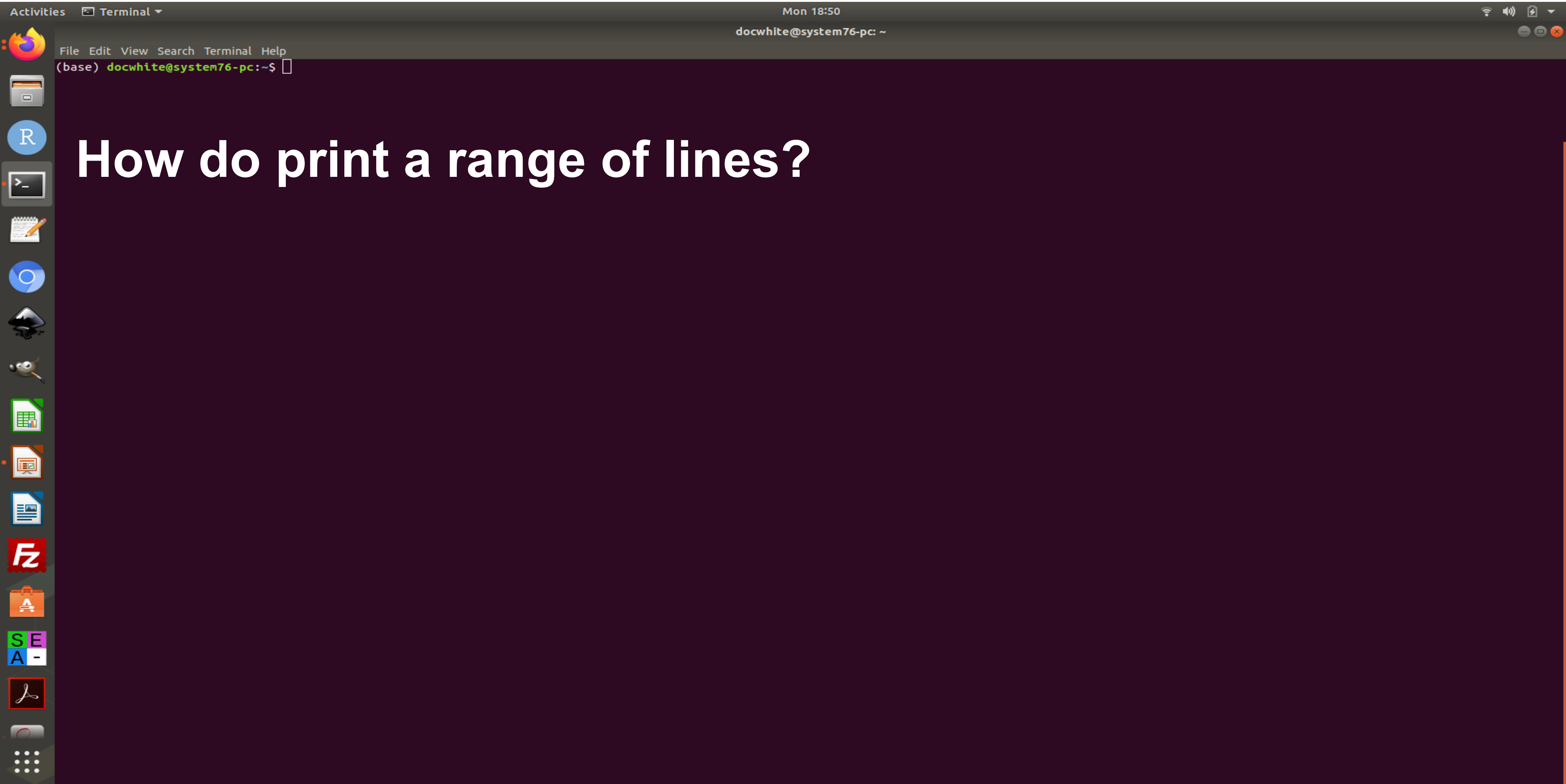
How do print the line number 1 only and grab david with color?

```
cat -n doppelganger_names.txt | sed -n '1p' | grep "david" --color
```

How do I do it with head/tail command?

```
cat -n doppelganger_names.txt | head -1 | tail +1 | grep "david" --color
```

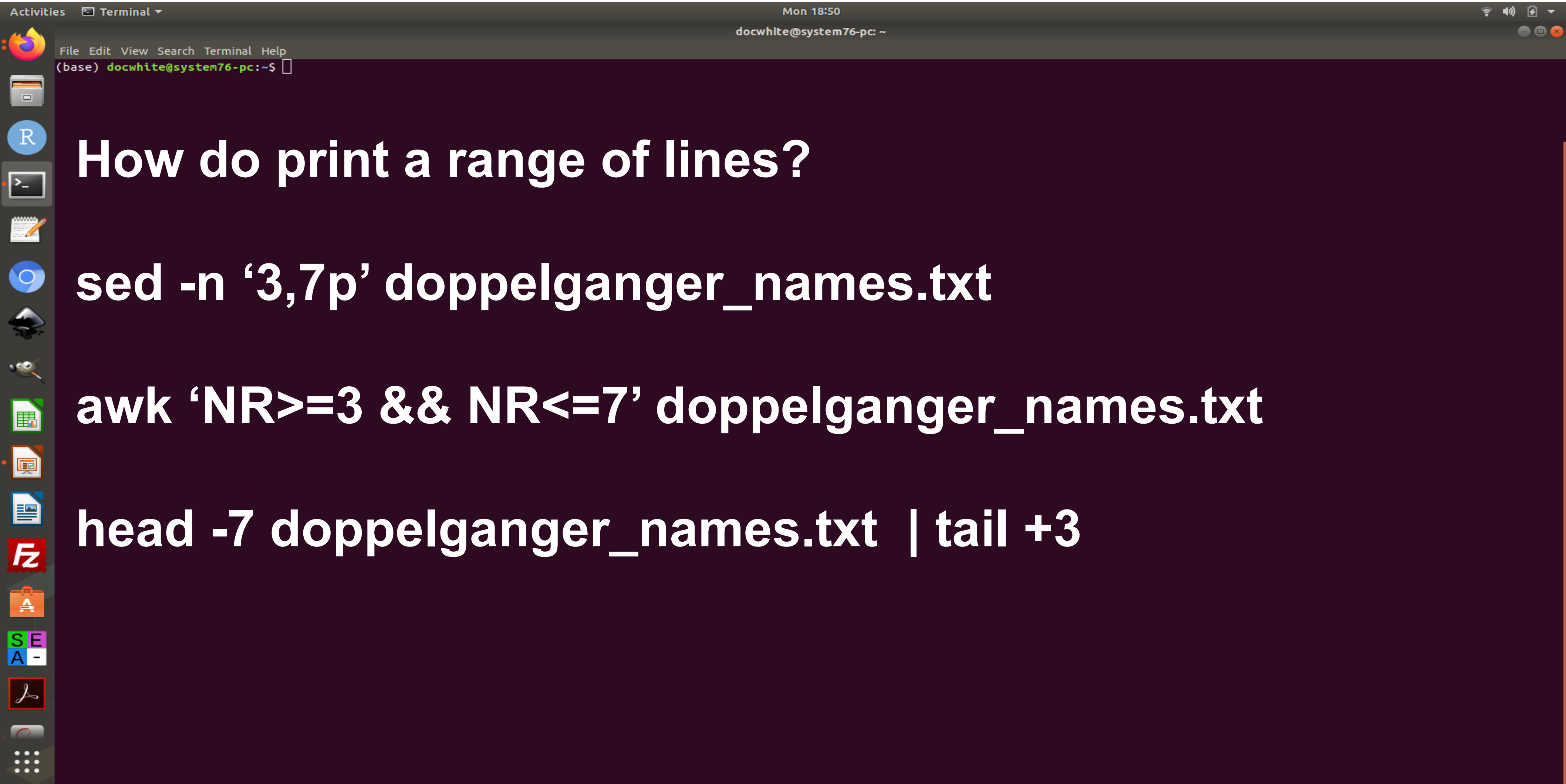
Specific lines



```
(base) docwhite@system76-pc:~$ 
```

How do print a range of lines?

Specific lines

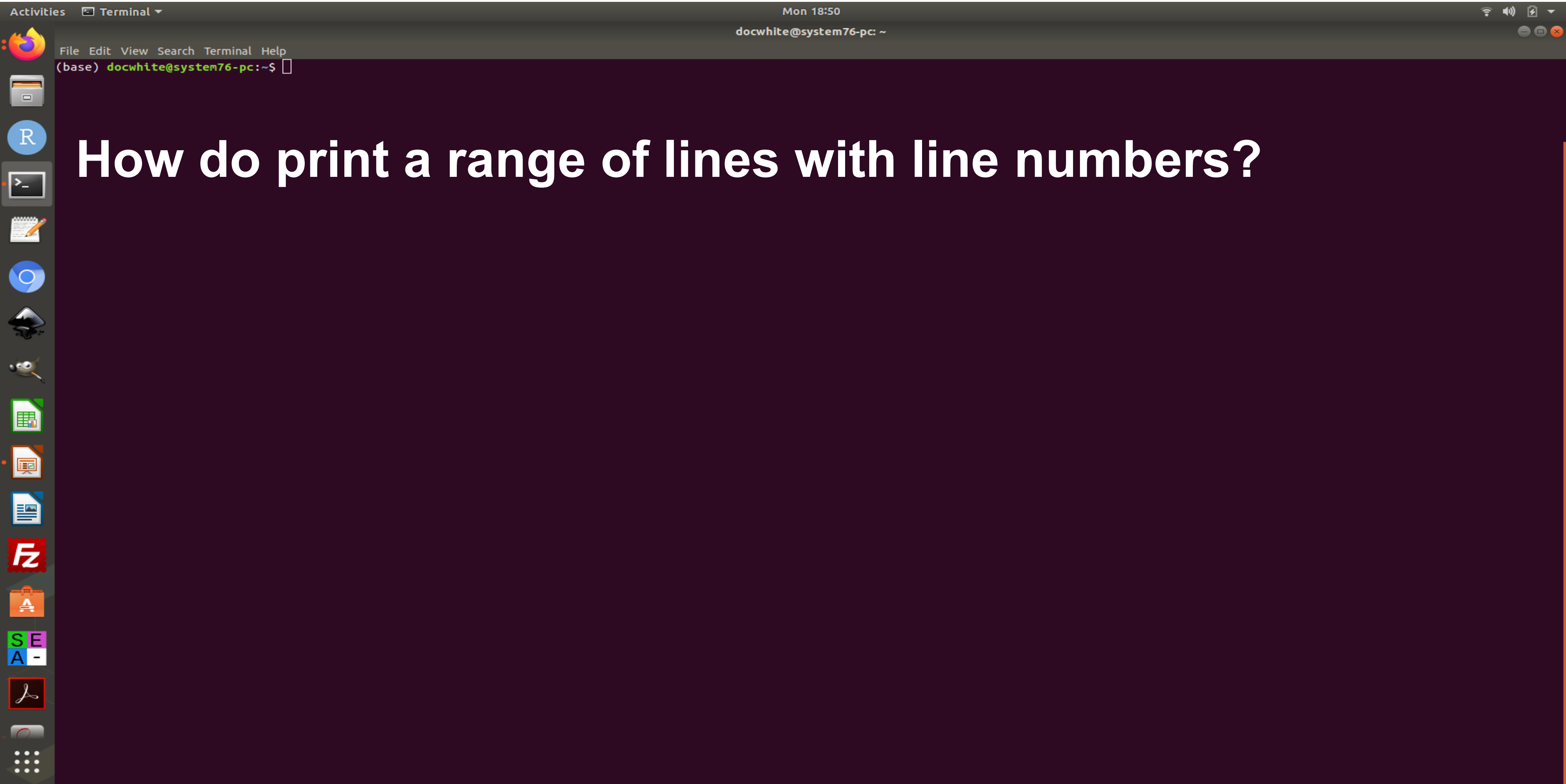


```
(base) docwhite@system76-pc:~$ 
```

How do print a range of lines?

```
sed -n ‘3,7p’ doppelganger_names.txt

awk ‘NR>=3 && NR<=7’ doppelganger_names.txt

head -7 doppelganger_names.txt  | tail +3
```

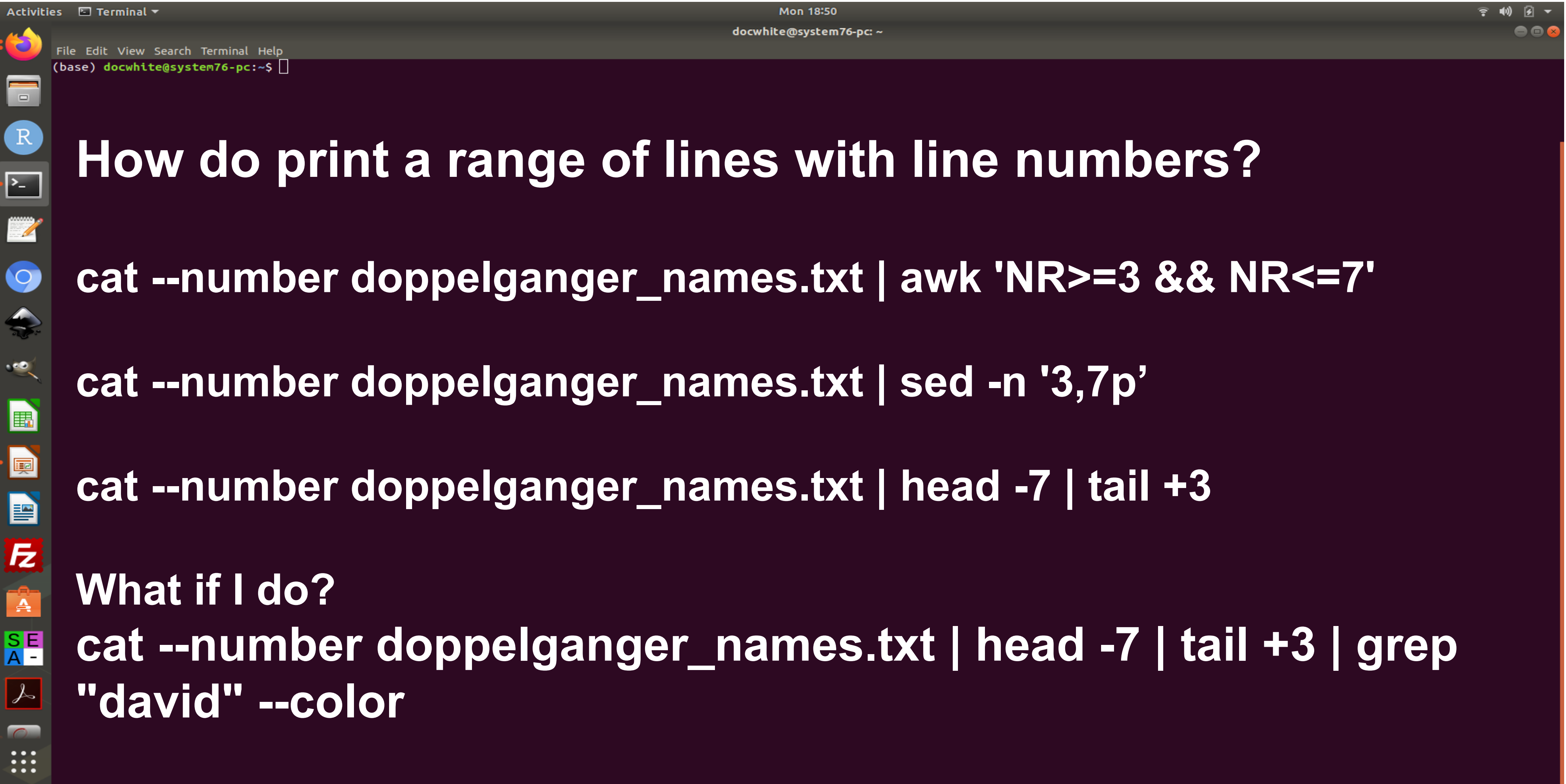
Specific lines



```
(base) docwhite@system76-pc:~$ 
```

How do print a range of lines with line numbers?

Specific lines



```
(base) docwhite@system76-pc:~$
```

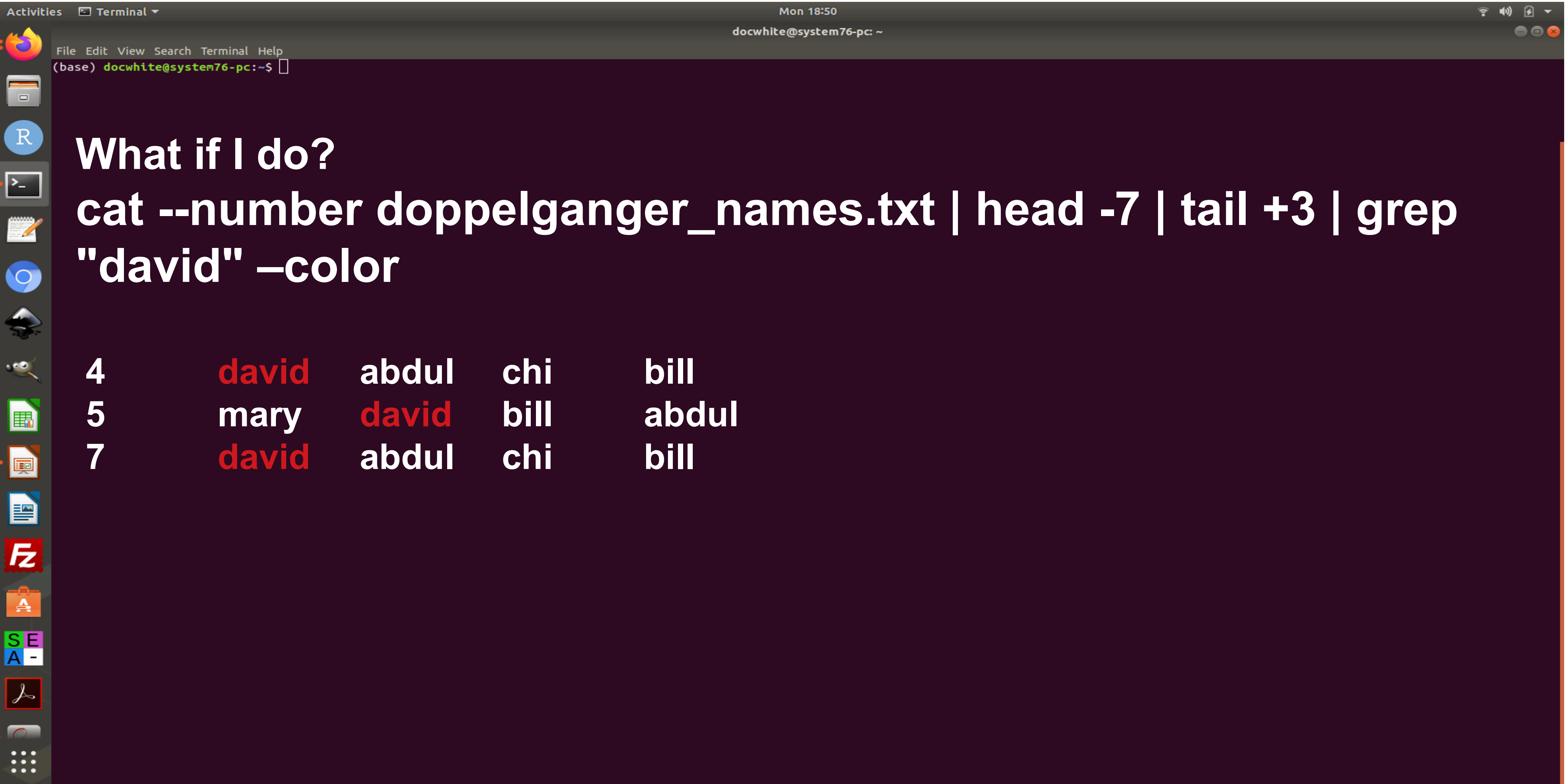
How do print a range of lines with line numbers?

```
cat --number doppelganger_names.txt | awk 'NR>=3 && NR<=7'

cat --number doppelganger_names.txt | sed -n '3,7p’

cat --number doppelganger_names.txt | head -7 | tail +3
```

What if I do?

```
cat --number doppelganger_names.txt | head -7 | tail +3 | grep
"david" --color
```

Specific lines



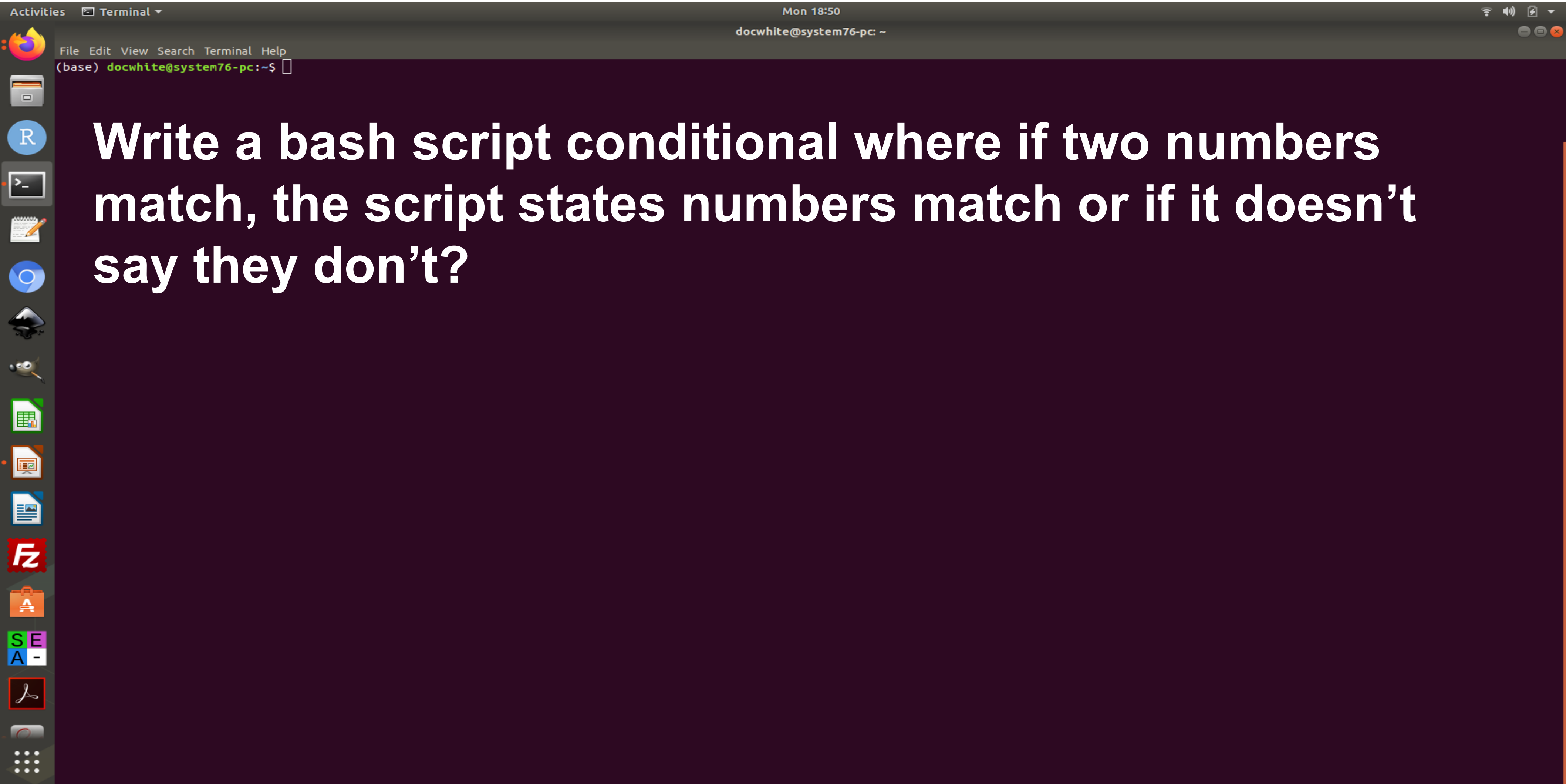
```
(base) docwhite@system76-pc:~$
```

What if I do?

```
cat --number doppelganger_names.txt | head -7 | tail +3 | grep
"david" –color
```

```
4       david   abdul   chi     bill
5       mary    david   bill    abdul
7       david   abdul   chi     bill
```

Write a bash script (conditionals)



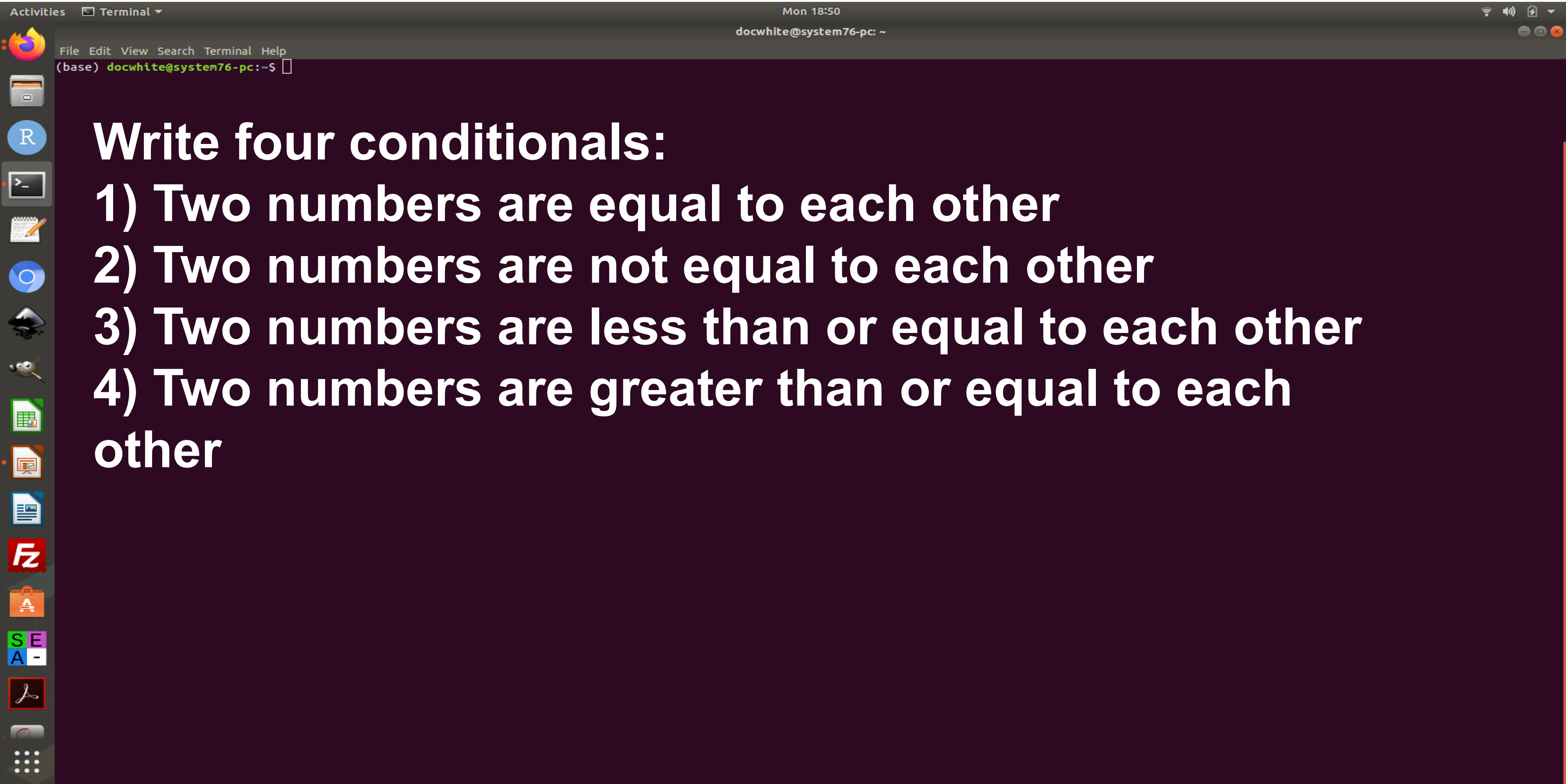
```
(base) docwhite@system76-pc:~$ 
```

Write a bash script conditional where if two numbers match, the script states numbers match or if it doesn’t say they don’t?

Write a bash script (conditionals)

```
#!/bin/bash

num1=$1
num2=$2

if [ $num1 -eq $num2 ]; then
    echo "the numbers match"
else
    echo "the numbers dont match"
fi
```

Write a bash script (conditionals)



```
(base) docwhite@system76-pc:~$
```

Write four conditionals:

1) Two numbers are equal to each other
2) Two numbers are not equal to each other
3) Two numbers are less than or equal to each other
4) Two numbers are greater than or equal to each other

Write a bash script (conditionals, first)

```bash
#!/bin/bash

num1=$1
num2=$2

if [ $num1 == $num2 ]; then
    echo "Are equal"
else
    echo "Not equal"
fi
```

Write a bash script (conditionals, second)

```
#!/bin/bash

num1=$1
num2=$2

if [ $num1 != $num2 ]; then
    echo "Not equal"
else
    echo "Are equal"
fi
```

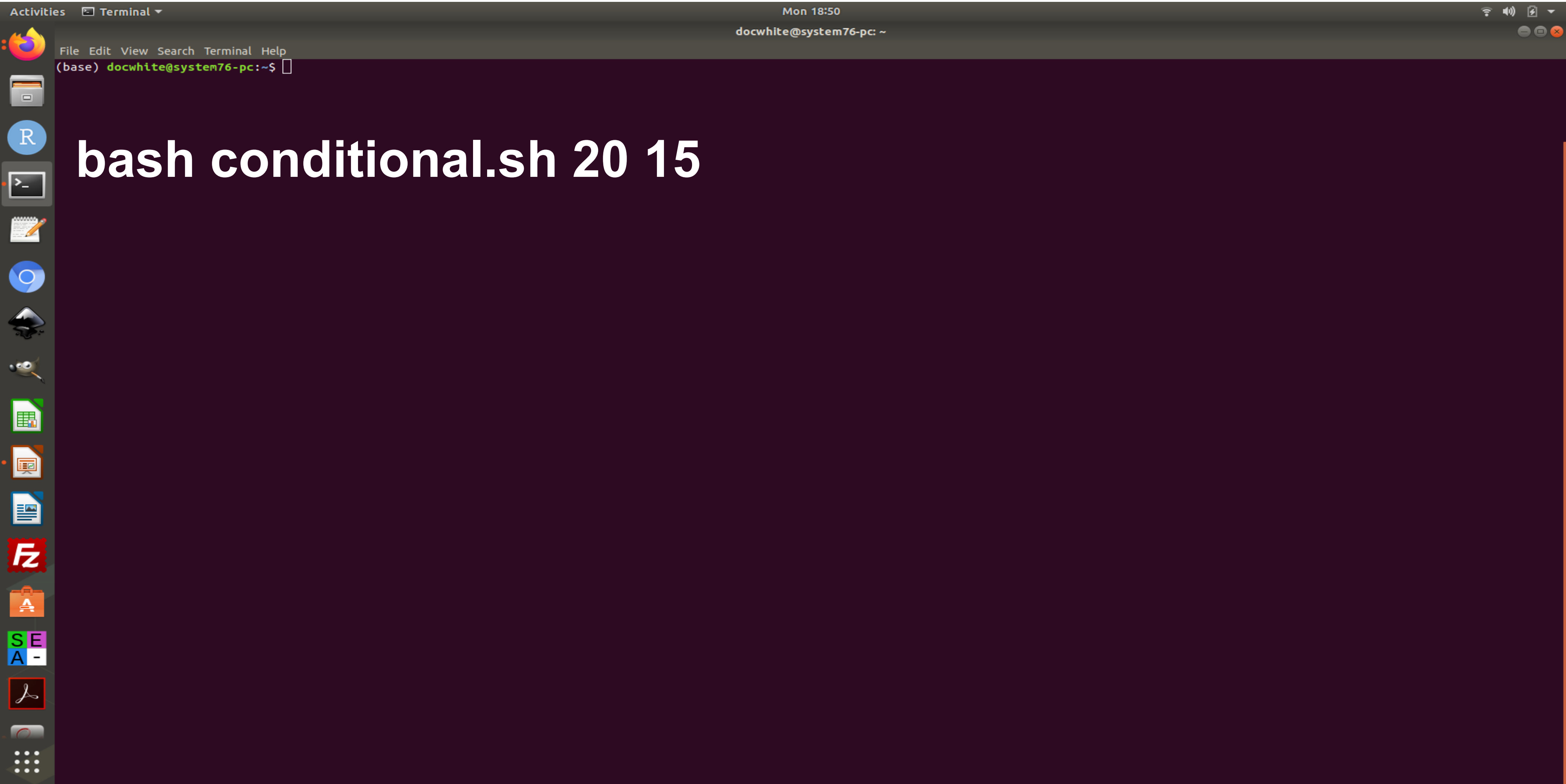
Write a bash script (conditionals, third)

```
#!/bin/bash

num1=$1
num2=$2

if [ $num1 -le $num2 ]; then
    echo "Less than or equal"
else
    echo "Not Less than or equal"
fi
```

Write a bash script (conditionals, third)

```bash
#!/bin/bash

num1=$1
num2=$2

if [ $num1 -ge $num2 ]; then
    echo "greater than or equal"
else
    echo "Not greater than or equal"
fi
```

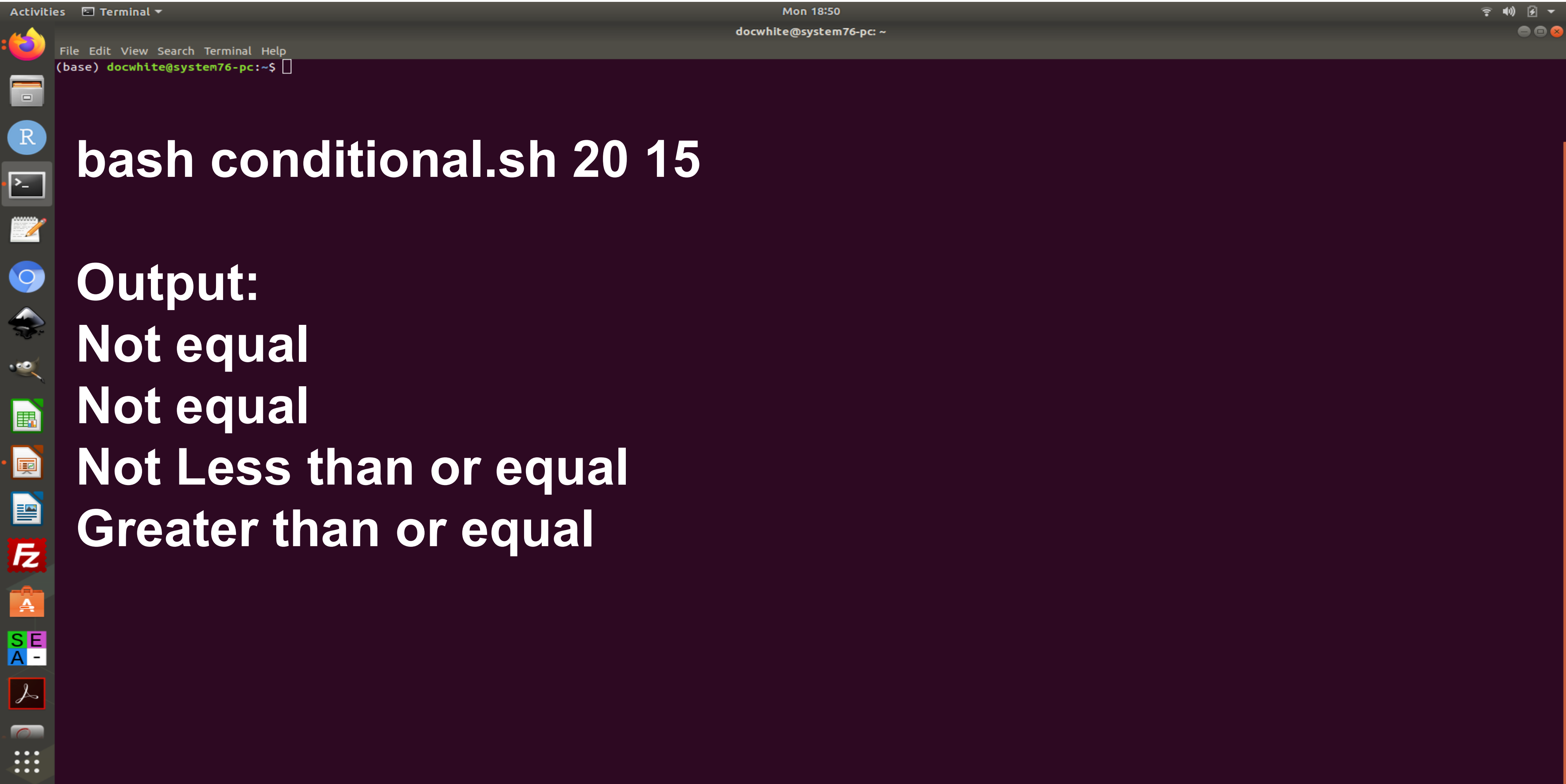
Conditional script



```
(base) docwhite@system76-pc:~$ 
```

```
bash conditional.sh 20 15
```

Conditional script



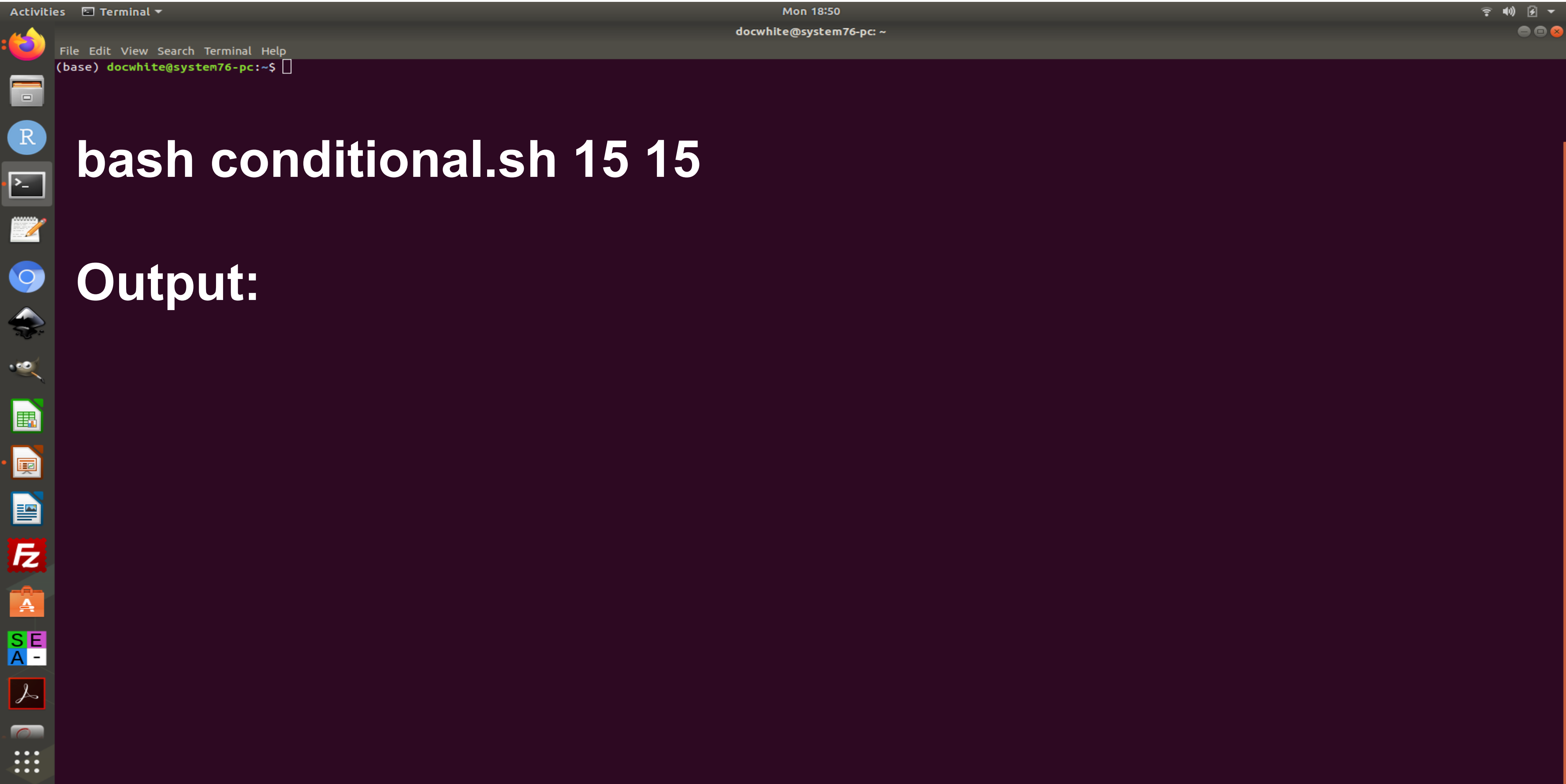
```
(base) docwhite@system76-pc:~$ 
```

```
bash conditional.sh 20 15
```

Output:

```
Not equal
Not equal
Not Less than or equal
Greater than or equal
```

Conditional script

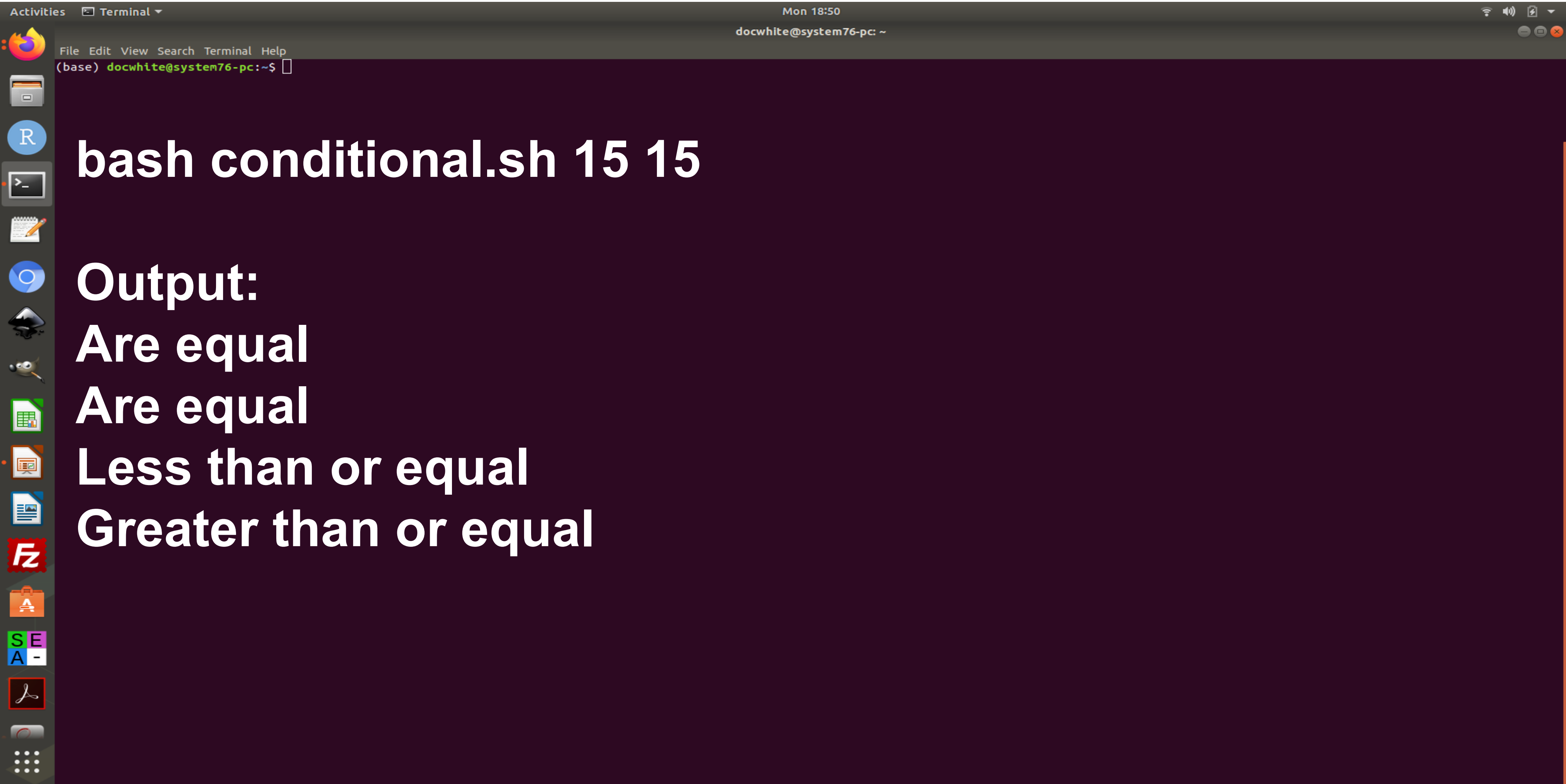


```
(base) docwhite@system76-pc:~$ 
```

bash conditional.sh 15 15

Output:

Conditional script



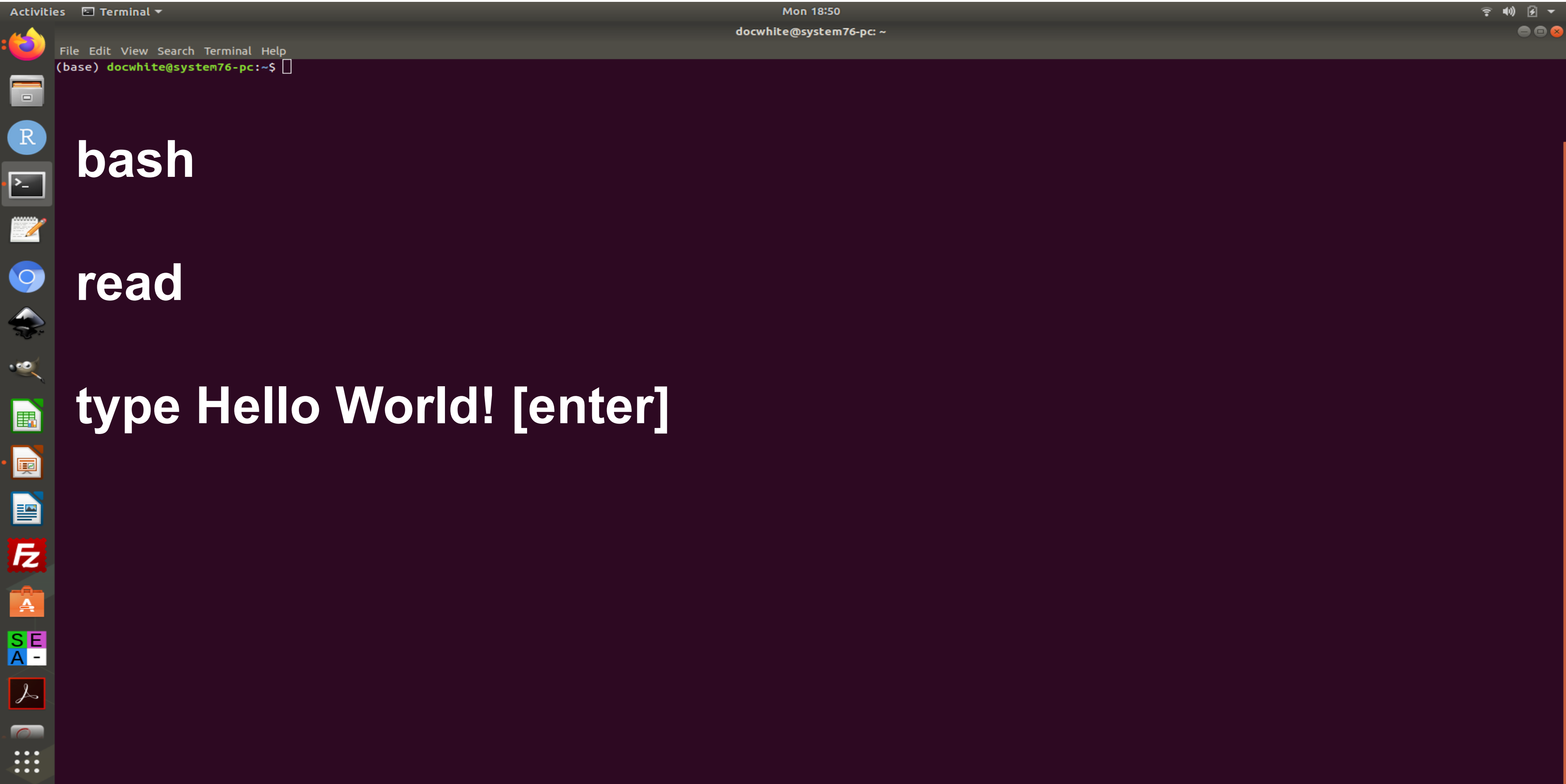
```
(base) docwhite@system76-pc:~$
```

```
bash conditional.sh 15 15
```

Output:

```
Are equal
Are equal
Less than or equal
Greater than or equal
```

Read command



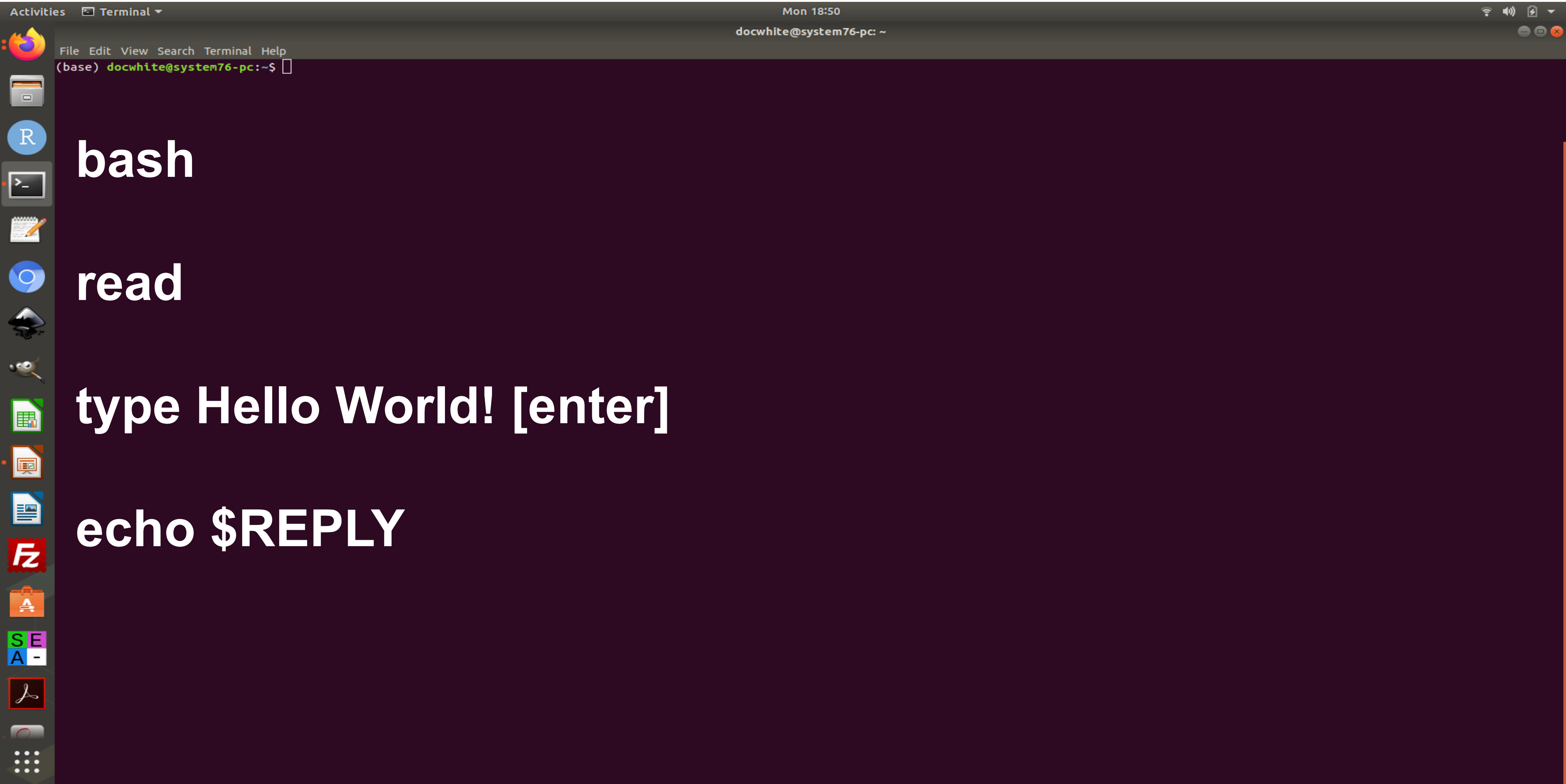
```
(base) docwhite@system76-pc:~$ 
```

bash

read

type Hello World! [enter]

Read command



```
(base) docwhite@system76-pc:~$ 
```

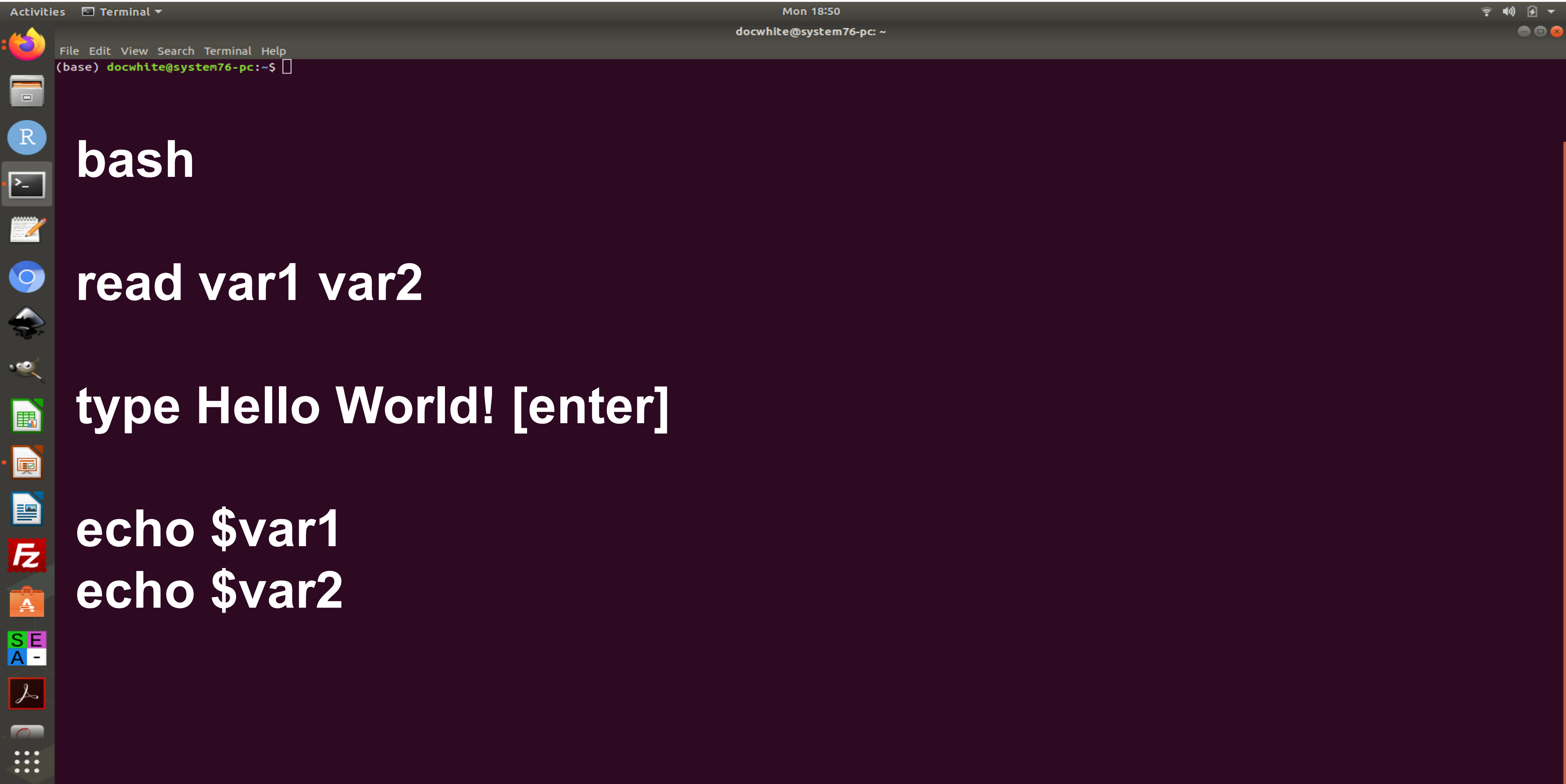
bash

read

type Hello World! [enter]

echo $REPLY

Read command



```
(base) docwhite@system76-pc:~$
```

bash

read var1 var2

type Hello World! [enter]

echo $var1
echo $var2

Read command

```
#!/bin/bash

echo "enter first number"
read num1

echo "enter second number"
read num2

if [ $num1 -ge $num2 ]; then
    echo "greater than or equal"
else
    echo "Not greater than or equal"
fi
```

BASH - for loop

```
for i in file.*;do
  command $i
done
```

BASH - for loop (C-style)

```
for ((i = 0 ; i < 100 ; i++)); do
  command $i
done
```

BASH - for loop

```
#!/bin/bash
for i in {1..5}
do
   echo "Welcome $i times"
done
```

```
#!/bin/bash
for (( c=1; c<=5; c++ ))
do
   echo "Welcome $i times"
done
```

BASH - for loop

Hashbang line

```
#!/bin/bash
for i in {1..5}
do
   echo "Welcome $i times"
done
```

```
#!/bin/bash
for (( c=1; c<=5; c++ ))
do
   echo "Welcome $i times"
done
```

BASH - for loop

Hashbang line

```
#!/bin/bash
for i in {1..5}
do
  echo "Welcome $i times"
done
```

```
#!/bin/bash
for (( c=1; c<=5; c++ ))
do
  echo "Welcome $i times"
done
```

for i in {1..5}: This line starts a for loop. Here's what each part does: for i in ...:
This defines a loop where the variable i will take on values from a specified range.
{1..5}: This is a brace expansion that generates a sequence of numbers from 1 to 5 (inclusive). It will be assigned each of these numbers in each iteration of the loop.

BASH - for loop

Hashbang line

```
#!/bin/bash
for i in {1..5}
do
  echo "Welcome $i times"
done
```

Begin loop

for i in {1..5}: This line starts a for loop. Here's what each part does: for i in ...:
This defines a loop where the variable i will take on values from a specified range.
{1..5}: This is a brace expansion that generates a sequence of numbers from 1 to 5 (inclusive). It will be assigned each of these numbers in each iteration of the loop.

```
#!/bin/bash
for (( c=1; c<=5; c++ ))
do
  echo "Welcome $i times"
done
```

BASH - for loop

Hashbang line

```
#!/bin/bash
for i in {1..5}
do
   echo "Welcome $i times"
done
```

Begin loop

End loop

```
#!/bin/bash
for (( c=1; c<=5; c++ ))
do
   echo "Welcome $i times"
done
```

for i in {1..5}: This line starts a for loop. Here's what each part does: for i in ...:
This defines a loop where the variable i will take on values from a specified range.
{1..5}: This is a brace expansion that generates a sequence of numbers from 1 to 5 (inclusive). It will be assigned each of these numbers in each iteration of the loop.

BASH - for loop

Hashbang line

```
#!/bin/bash
for i in {1..5}
do
  echo "Welcome $i times"
done
```

Begin loop

End loop

```
#!/bin/bash
for (( c=1; c<=5; c++ ))
do
  echo "Welcome $i times"
done
```

for i in {1..5}: This line starts a for loop. Here's what each part does: for i in ...:
This defines a loop where the variable i will take on values from a specified range.
{1..5}: This is a brace expansion that generates a sequence of numbers from 1 to 5 (inclusive). It will be assigned each of these numbers in each iteration of the loop.

for ((c=1; c<=5; c++)): This line starts a for loop. Here's what each part does:

((...)): This is an arithmetic evaluation construct in Bash.

c=1: Initializes a variable c to 1. This variable will be used as the loop counter.

c<=5: This is the condition for the loop. The loop will continue executing as long as c is less than or equal to 5.

c++: This is the increment statement. It increases the value of c by 1 in each iteration.

BASH - for loop

```bash
#!/bin/bash
for (( c=1; c<=5; c++ ))
do
  echo "Welcome $i times"
done
```

```c
#include <stdio.h>

int main() {
    for (int c = 1; c <= 5; c++) {
        printf("Welcome %d times\n", c);
    }

    return 0;
}
```

C code

BASH - for loop (C vs. C++)

```
#include <stdio.h>

int main() {
    for (int c = 1; c <= 5; c++) {
        printf("Welcome %d times\n", c);
    }

    return 0;
}
```

C code

```
#include <iostream>

int main() {
    for (int c = 1; c <= 5; c++) {
        std::cout << "Welcome " << c
<< " times" << std::endl;
    }

    return 0;
}
```

C++ code

BASH - for loop (C vs. C++)

```
#include <stdio.h>

int main() {
    for (int c = 1; c <= 5; c++) {
        printf("Welcome %d times\n", c);
    }

    return 0;
}
```

C code

```
#include <iostream>

int main() {
    for (int c = 1; c <= 5; c++) {
        std::cout << "Welcome " << c
<< " times" << std::endl;
    }

    return 0;
}
```

C++ code

c = 1 is the Variable

c <= 5 is Conditional

c++ is Increment statement

For loop (Java)

```
for (i = 0 ; i < 100 ; i++){
    command (i);
}
```

for loop (Python)

```
for x in file:
        command
(x)
```

for loop (Python)

```
#!/bin/usr/python3

for c in range(1, 6):
    print(f"Welcome {c} times")
```

for loop (Rust)

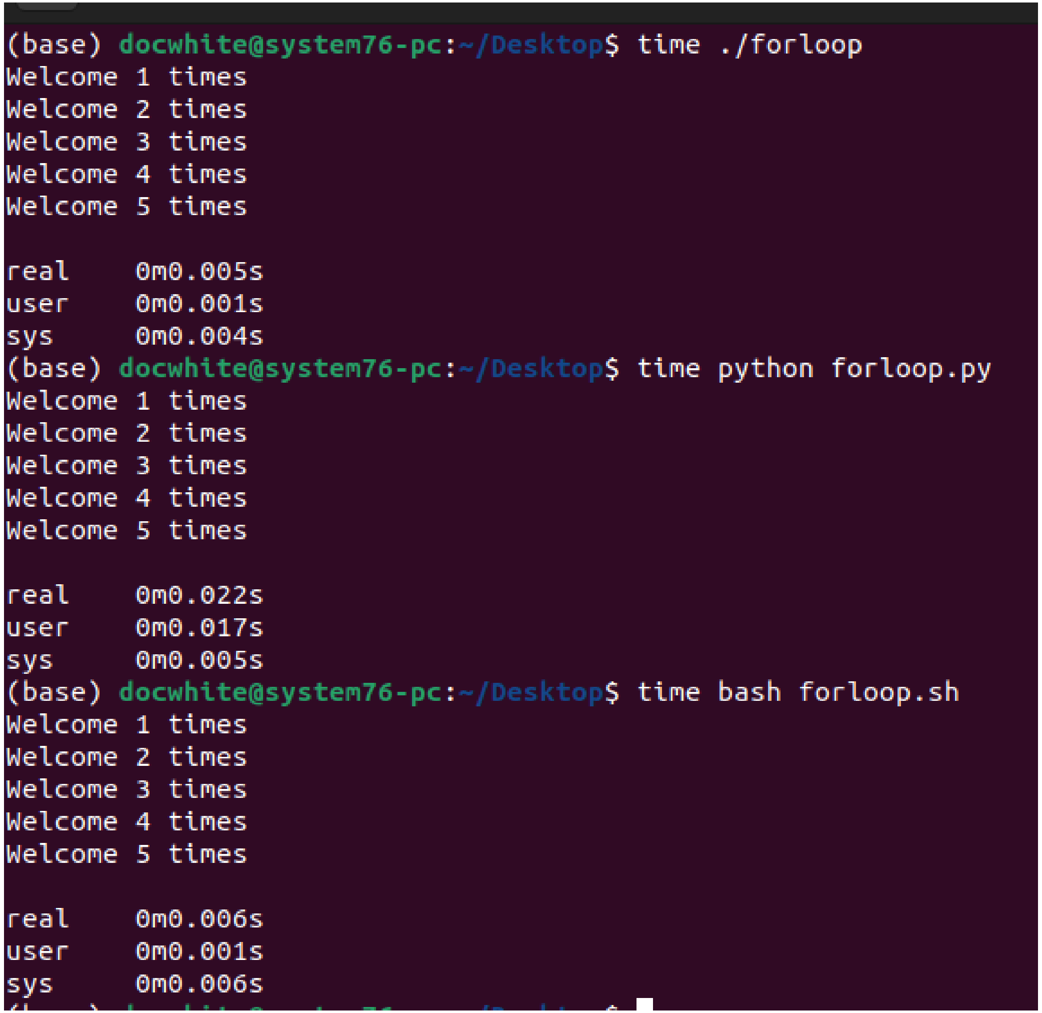
```
for element in iterable {
    // Code to be executed for each element
}
```

for loop (Rust)

```
fn main() {
    for c in 1..=5 {
        println!("Welcome {} times", c);
    }
}
```

Compile -> rustc forloop.rs

for loop - Rust vs. Python vs. Bash



```
(base) docwhite@system76-pc:~/Desktop$ time ./forloop
Welcome 1 times
Welcome 2 times
Welcome 3 times
Welcome 4 times
Welcome 5 times

real	0m0.005s
user	0m0.001s
sys	0m0.004s
(base) docwhite@system76-pc:~/Desktop$ time python forloop.py
Welcome 1 times
Welcome 2 times
Welcome 3 times
Welcome 4 times
Welcome 5 times

real	0m0.022s
user	0m0.017s
sys	0m0.005s
(base) docwhite@system76-pc:~/Desktop$ time bash forloop.sh
Welcome 1 times
Welcome 2 times
Welcome 3 times
Welcome 4 times
Welcome 5 times

real	0m0.006s
user	0m0.001s
sys	0m0.006s
(base) docwhite@system76-pc:~/Desktop$
```

Quiz 10

- On canvas now

Bonus 10

- Write a bash script that prints the working directory, counts all the sequences within a fasta files within the working directory, and prints the first five lines of the file into std_out.txt?