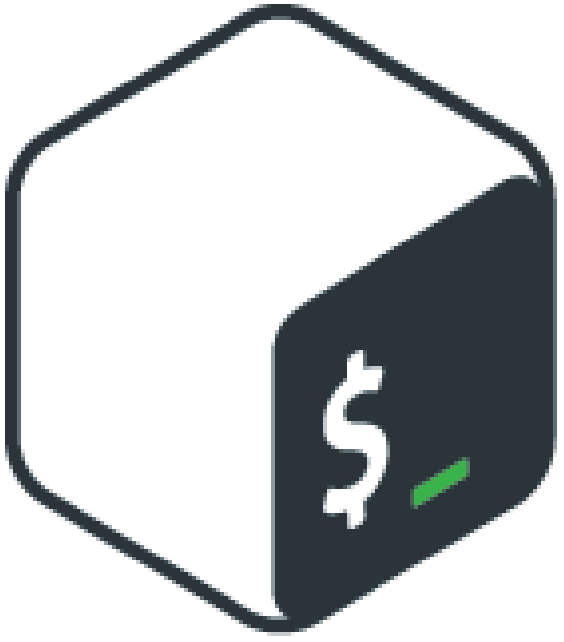
BINF2111 – Introduction to Bioinformatics Computing

BASH 101 – Bash around and find out?



BASH

THE BOURNE-AGAIN SHELL

Richard Allen White III, PhD
RAW Lab

Lecture 7 – Tuesday Sep 13th, 2022

Learning Objectives

- When are labs and quizzes due

- Review quiz and bonus

- Shell vs. bash (bash variants)

- Bash

- Bash variables

- Quiz 7

Bonus 6

In the doppelganger_names.txt count how many times the name ‘chi’ is left to the name ‘bill’

Using grep only command:

Using grep with printf command:

Only awk:

Bonus 5

In the doppelganger_names.txt count how many times the name ‘chi’ is left to the name ‘bill’

Using grep only command:

```
grep -oE 'chi'$'\t''bill' doppelganger_names.txt | wc -l
```

Using grep with printf command:

```
grep -oE "chi$(printf '\t')bill" doppelganger_names.txt | wc -l
```

Only awk:

```
awk '/chi\tbill/' doppelganger_names.txt | wc -l
```

Bonus 6

```
grep -oE 'chi'$'\t''bill' doppelganger_names.txt
grep -oE "chi$(printf '\t')bill" doppelganger_names.txt

chi	bill
chi	bill
chi	bill
chi	bill

awk '/chi\tbill/' doppelganger_names.txt

david abdul chi bill
david abdul chi bill
david abdul chi bill
david abdul chi bill
```

Quiz answers - printf

printf '#!/bin/bash\n \n#Written by RAWIII\n' >test.txt

Produces this output

more test.txt

#!/bin/bash
#Written by RAWIII

T or F

Quiz answers - printf

printf '#!/bin/bash\n \n#Written by RAWIII\n' >test.txt

Produces this output

more test.txt

#!/bin/bash
#Written by RAWIII

T or F

Quiz answers - printf

printf '#!/bin/bash\n \n#Written by RAWIII\n' >test.txt

Produces this output

more test.txt

#!/bin/bash

#Written by RAWIII

T or F

Quiz answers - printf

printf '#!/bin/bash\n \n#Written by RAWIII\n' >test.txt

Produces this output

more test.txt

#!/bin/bash

#Written by RAWIII

T or F

Quiz answers - empty lines

```
grep -v -e '^$$' file.txt
```

Deletes all empty lines?

T or F

Quiz answers - empty lines

```
grep -v -e '^$$' file.txt
```

Deletes all empty lines?

T or F

Quiz answers - empty lines

```
grep -v -e '^$$' file.txt
```

Deletes all empty lines?

grep: `grep -v -e '^$' file`
awk: `awk '!/^$/' file`

Lab Two most missed questions

Write a sed command that deletes lines with last word being rat?

more rat.txt
Rat Cat Mat Stat rat
rat cat maT stat cat
cat rat mat STat bat

Lab Two most missed questions

Write a sed command that deletes lines with last word being rat?

more rat.txt
Rat Cat Mat Stat rat
rat cat maT stat cat
cat rat mat STat bat

sed '/rat/d' test.txt or sed '/[Rr]at/d' test.txt

Lab Two most missed question

Write a sed command that deletes lines with last word being rat?

more rat.txt
Rat Cat Mat Stat rat
rat cat maT stat cat
cat rat mat STat bat

sed '/rat/d' test.txt or sed '/[Rr]at/d' test.txt

Lab Two most missed questions

Count the numbers of 'A', 'T', 'G', 'C' using multiple or single grep commands in example.fasta?

Lab Two most missed questions

Count the numbers of 'A', 'T', 'G', 'C' using multiple or single grep commands in example.fasta?

egrep -o 'A|T|G|C' example.fasta | wc -l
grep -oE 'A|T|G|C' example.fasta | wc -l

168?

Lab Two most missed questions

Count the numbers of 'A', 'T', 'G', 'C' using multiple or single grep commands in example.fasta?

```
egrep -o 'A|T|G|C' example.fasta | wc -l
grep -oE 'A|T|G|C' example.fasta | wc -l
```

168?

THIS WOULD BE WRONG! But, why?

Lab Two most missed questions

>chr1_geneA
CTAAGGCTATCTTGACAACTGACT
>chr1_geneB
CTAAGGCTATGTTGGCAACTGACT
>chr1_geneC
CTAAGGCTACCTTGACAACTGACT
>chr1_geneD
AAAAGGCTATCTTGACAACTGACT
>chr1_geneX
CTAAGGCTATCTTGATTTCTGACT
>chr1_geneY
GGGGGGCTATCTTGACAACTGACT
>chr1_geneZ
CTAAGGCTATCNNGACAACTGACT

Lab Two most missed questions

>chr1_gene**A**
CTAAGGCTATCTTGACAACTGACT
>chr1_geneB
CTAAGGCTATGTTGGCAACTGACT
>chr1_gene**C**
CTAAGGCTACCTTGACAACTGACT
>chr1_geneD
AAAAGGCTATCTTGACAACTGACT
>chr1_geneX
CTAAGGCTATCTTGATTTCTGACT
>chr1_geneY
GGGGGGCTATCTTGACAACTGACT
>chr1_geneZ
CTAAGGCTATCNNGACAACTGACT

Lab Two most missed questions

>chr1_gene**A**
CTAAGGCTATCTTGACAACTGACT
>chr1_geneB
CTAAGGCTATGTTGGCAACTGACT
>chr1_gene**C**
CTAAGGCTACCTTGACAACTGACT
>chr1_geneD
AAAAGGCTATCTTGACAACTGACT
>chr1_geneX
CTAAGGCTATCTTGATTTCTGACT
>chr1_geneY
GGGGGGCTATCTTGACAACTGACT
>chr1_geneZ
CTAAGGCTATCNNGACAACTGACT

Write a command where you could fix this?

Lab Two most missed questions

>chr1_gene**A**
CTAAGGCTATCTTGACAACTGACT
>chr1_geneB
CTAAGGCTATGTTGGCAACTGACT
>chr1_gene**C**
CTAAGGCTACCTTGACAACTGACT
>chr1_geneD
AAAAGGCTATCTTGACAACTGACT
>chr1_geneX
CTAAGGCTATCTTGATTTCTGACT
>chr1_geneY
GGGGGGCTATCTTGACAACTGACT
>chr1_geneZ
CTAAGGCTATCNNGACAACTGACT

sed 's/geneA/geneX2/g' example.fasta | sed ‘s/geneC/geneX3/g’ | egrep -o 'A|T|C|G' | wc -l

Lab Two most missed questions

>chr1_gene**A**
CTAAGGCTATCTTGACAACTGACT
>chr1_geneB
CTAAGGCTATGTTGGCAACTGACT
>chr1_gene**C**
CTAAGGCTACCTTGACAACTGACT
>chr1_geneD
AAAAGGCTATCTTGACAACTGACT
>chr1_geneX
CTAAGGCTATCTTGATTTCTGACT
>chr1_geneY
GGGGGGCTATCTTGACAACTGACT
>chr1_geneZ
CTAAGGCTATCNNGACAACTGACT

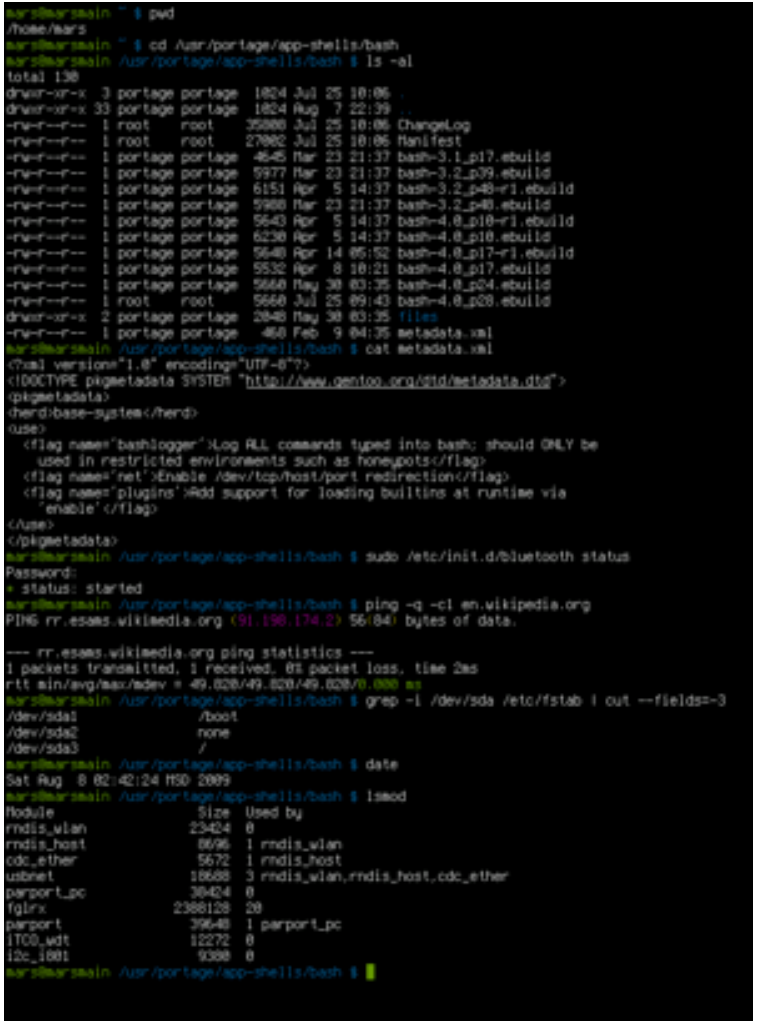
Always check your data! ;-)

Bourne shell (sh)

The Bourne shell (sh) is a shell command-line interpreter for computer operating systems.

The Bourne shell was the default shell for Version 7 Unix. Unix-like systems continue to have /bin/sh —which will be the Bourne shell, or a symbolic link or hard link to a compatible
shell even when other shells are used by most users.

Developed by Stephen Bourne at Bell Labs, it was a replacement for the Thompson shell, whose executable file had the same name—sh. It was released in 1979 in the Version 7 Unix release distributed to colleges and universities.



Variants of shell

C shell: Bill Joy, the author of the C shell, criticized the Bourne shell as being unfriendly for interactive use a task for which Stephen Bourne himself acknowledged C shell's superiority. Tom Christiansen also criticized C shell as being unsuitable for scripting and programming.

Korn shell: The Korn shell (ksh) written by David Korn based on the original Bourne Shell source code was a middle road between the Bourne shell and the C shell.

Z shell: developed by Paul Falstad in 1990, is an extended Bourne shell with a large number of improvements, including some features of Bash, ksh, and tcsh.

Schily Bourne Shell: Jörg Schilling's Schily-Tools includes three Bourne Shell derivatives

rc: rc shell was created at Bell Labs by Tom Duff as a replacement for sh for Version 10 Unix.

Shell	String processing	Alternation	Regrex	Pattern matching	Globbing qualifiers
Bourne 77	?	No	No	Yes	No
Bourne shell	Partial	No	No	Yes	No
POSIX shell	Partial	No	No	Yes	No
bash (v4.0)	Partial	Yes	Yes	Yes	No
csh	Yes	Yes	No	Yes	No
tcsh	Yes	Yes	Yes	Yes	No
Scsh	?	?	Yes	Yes	No
ksh	Partial	Yes	Yes	Yes	No
pdksh	?	Yes	No	Yes	No
zsh	Yes	Yes	Yes	Yes	Yes
ash	?	?	No	Yes	No

BASH

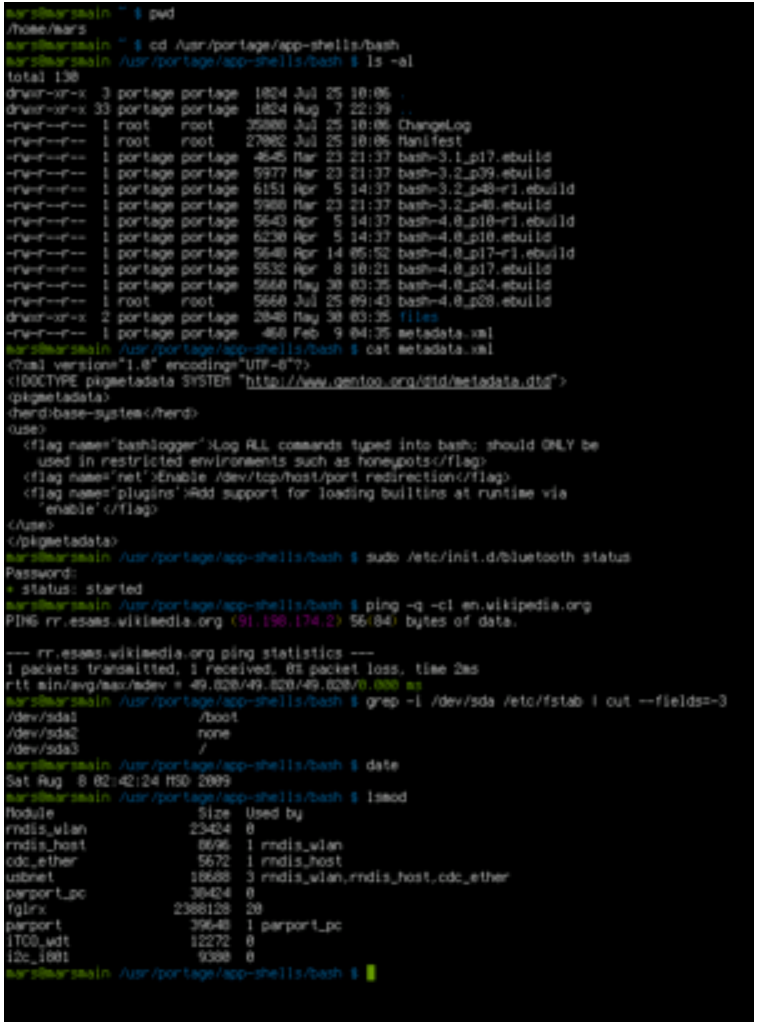
Bash is a Unix shell and command language written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell.

First released in 1989 it has been used as the default login shell for most Linux distributions.

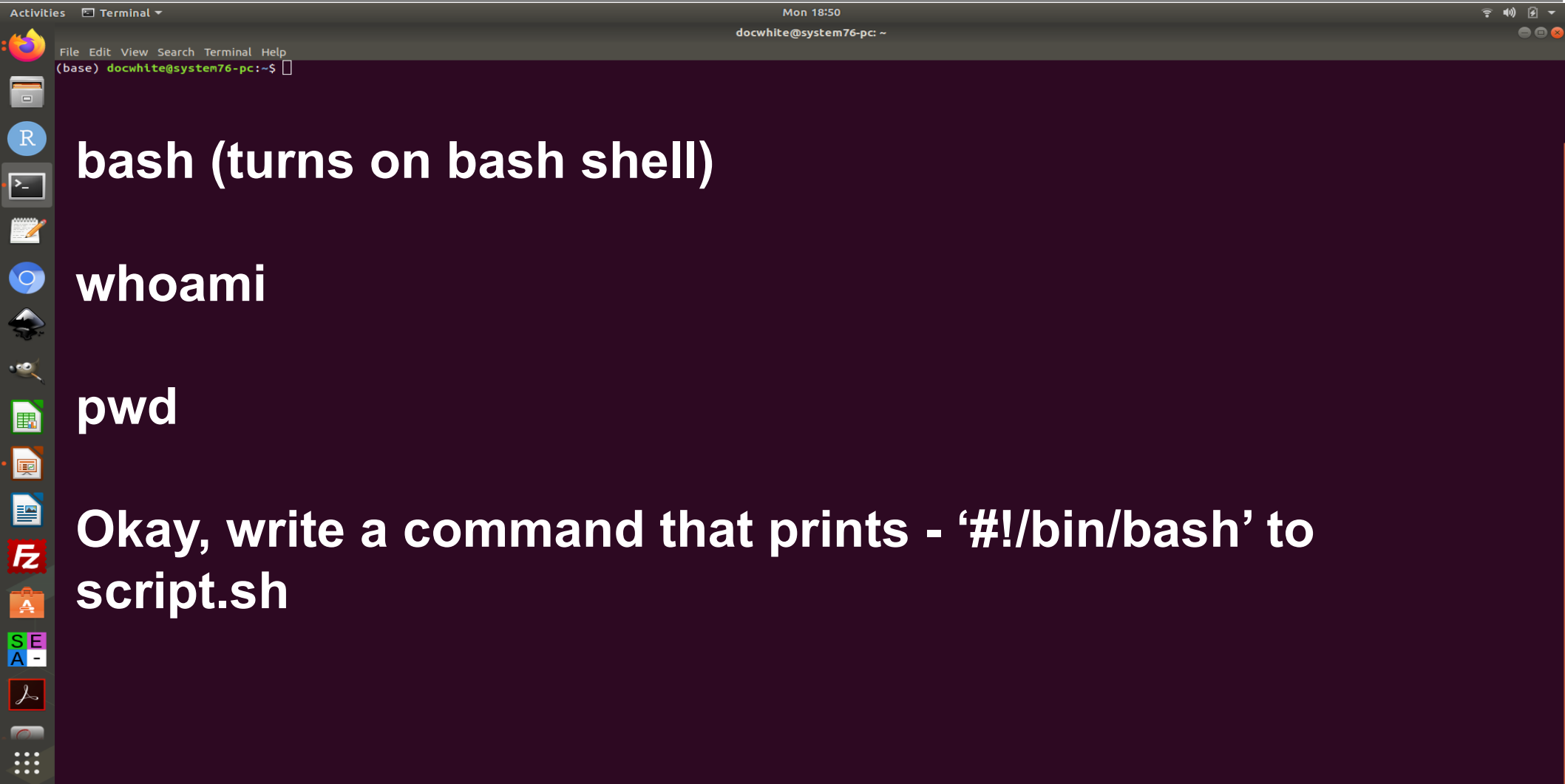
A version is also available for Windows 10 via the Windows Subsystem for Linux.

Bash was also the default shell in all versions of Apple macOS prior to the 2019 release of macOS Catalina, which changed the default shell to zsh.

The shell's name is an acronym for Bourne Again Shell, a pun on the name of the Bourne shell that it replaces and the notion of being "born again".

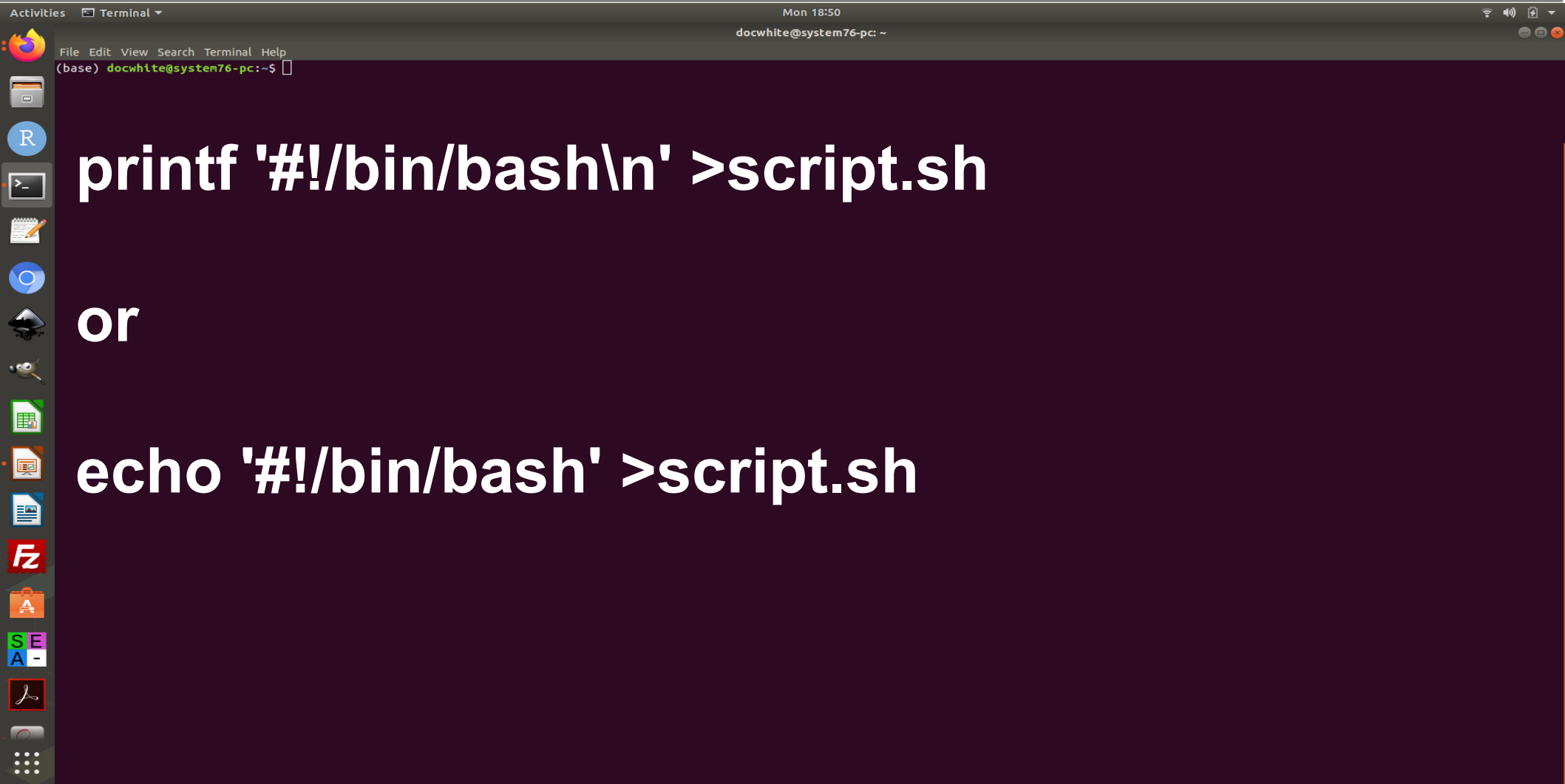


BASH examples

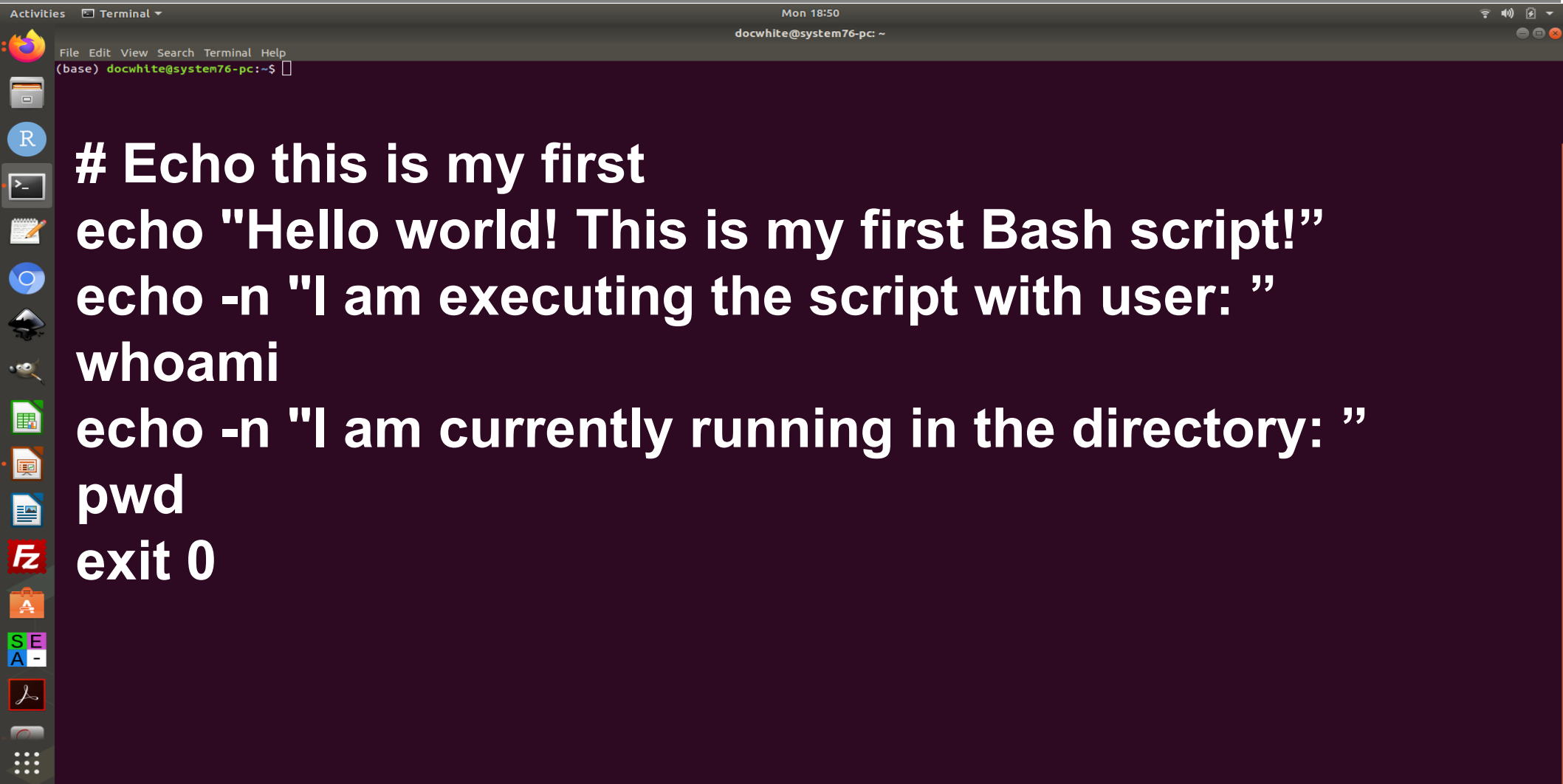


BASH examples

BASH examples

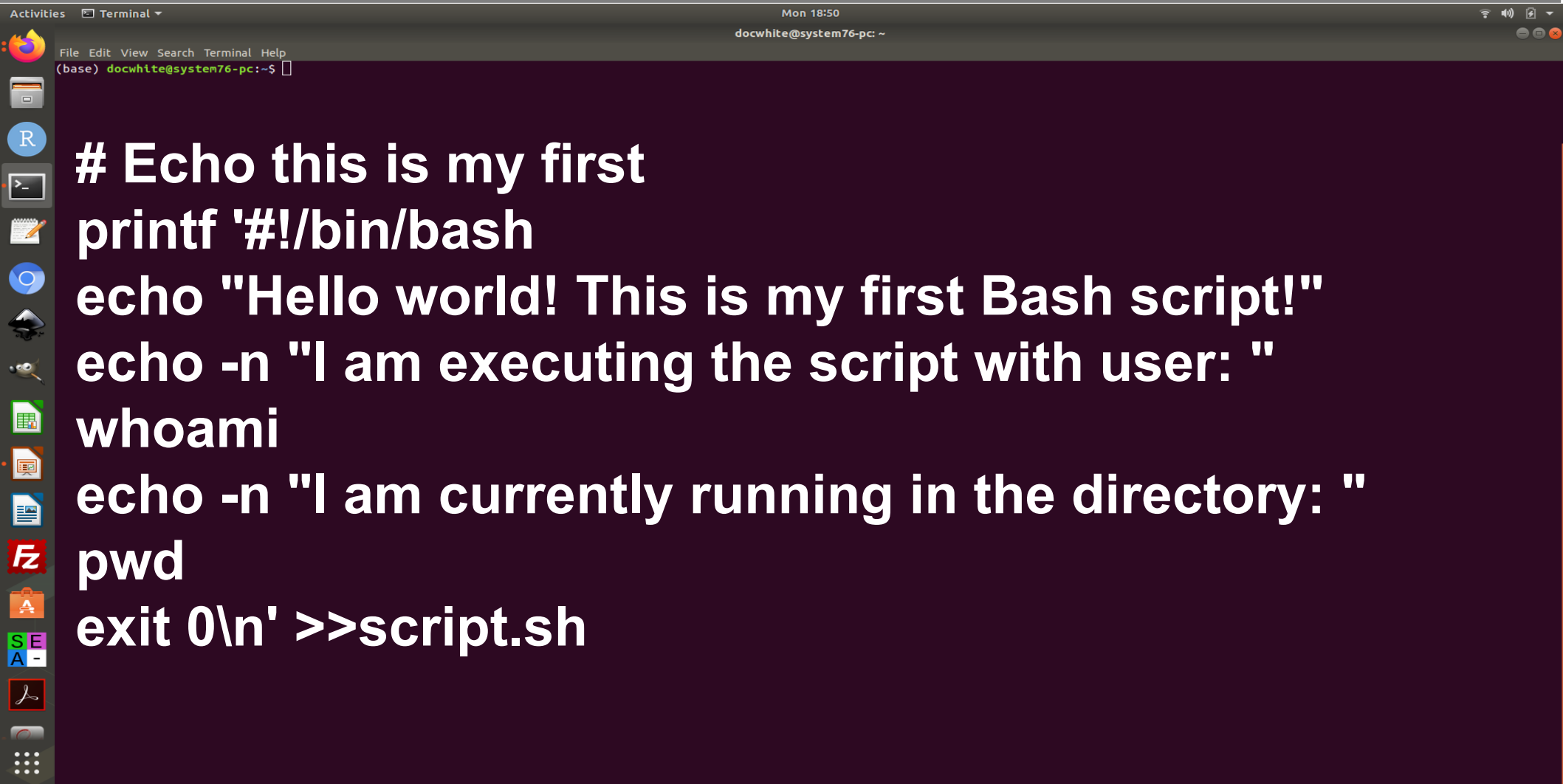


BASH examples

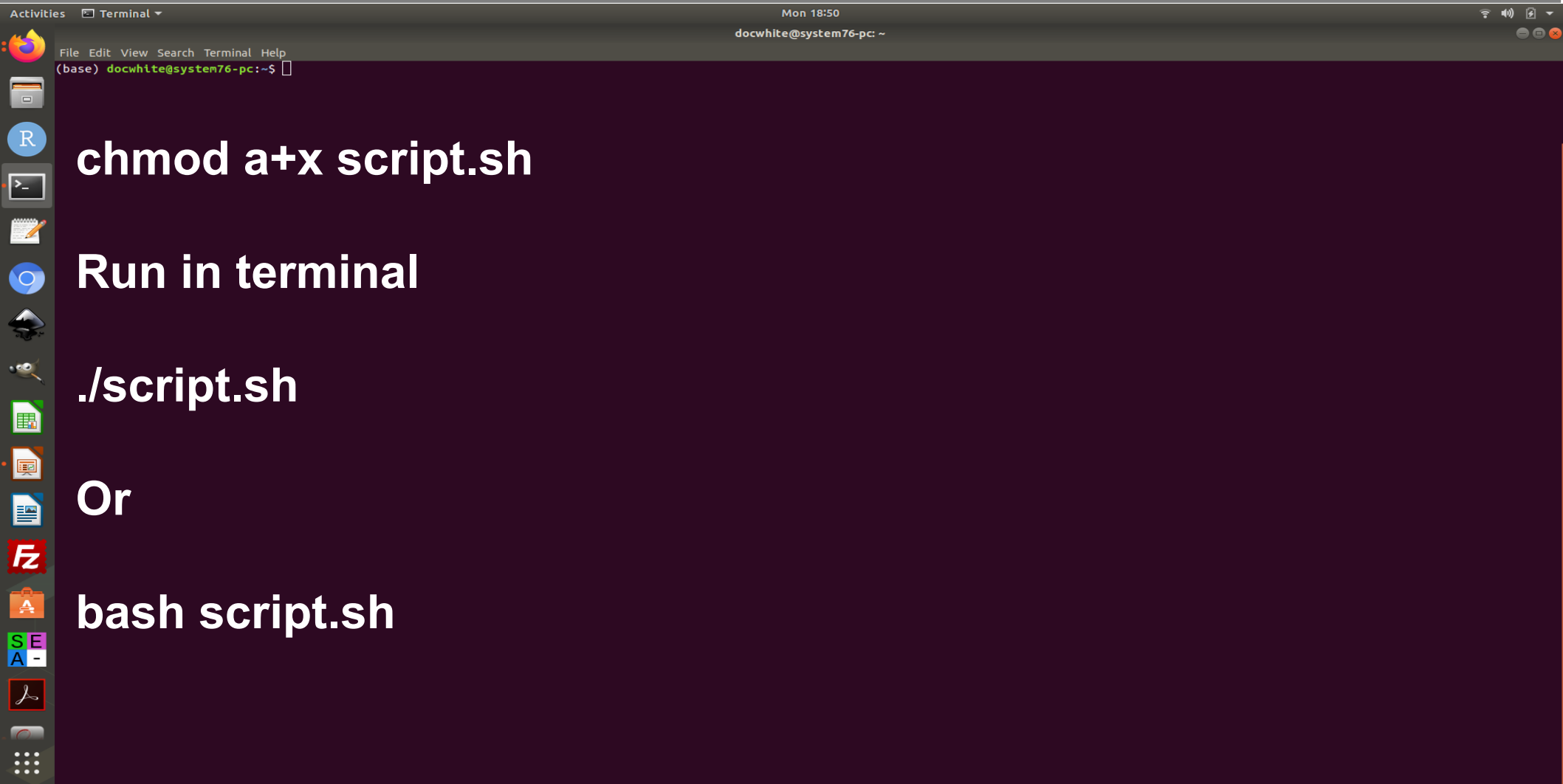



```
# Echo this is my first
echo "Hello world! This is my first Bash script!”
echo -n "I am executing the script with user: ”
whoami
echo -n "I am currently running in the directory: ”
pwd
exit 0
```

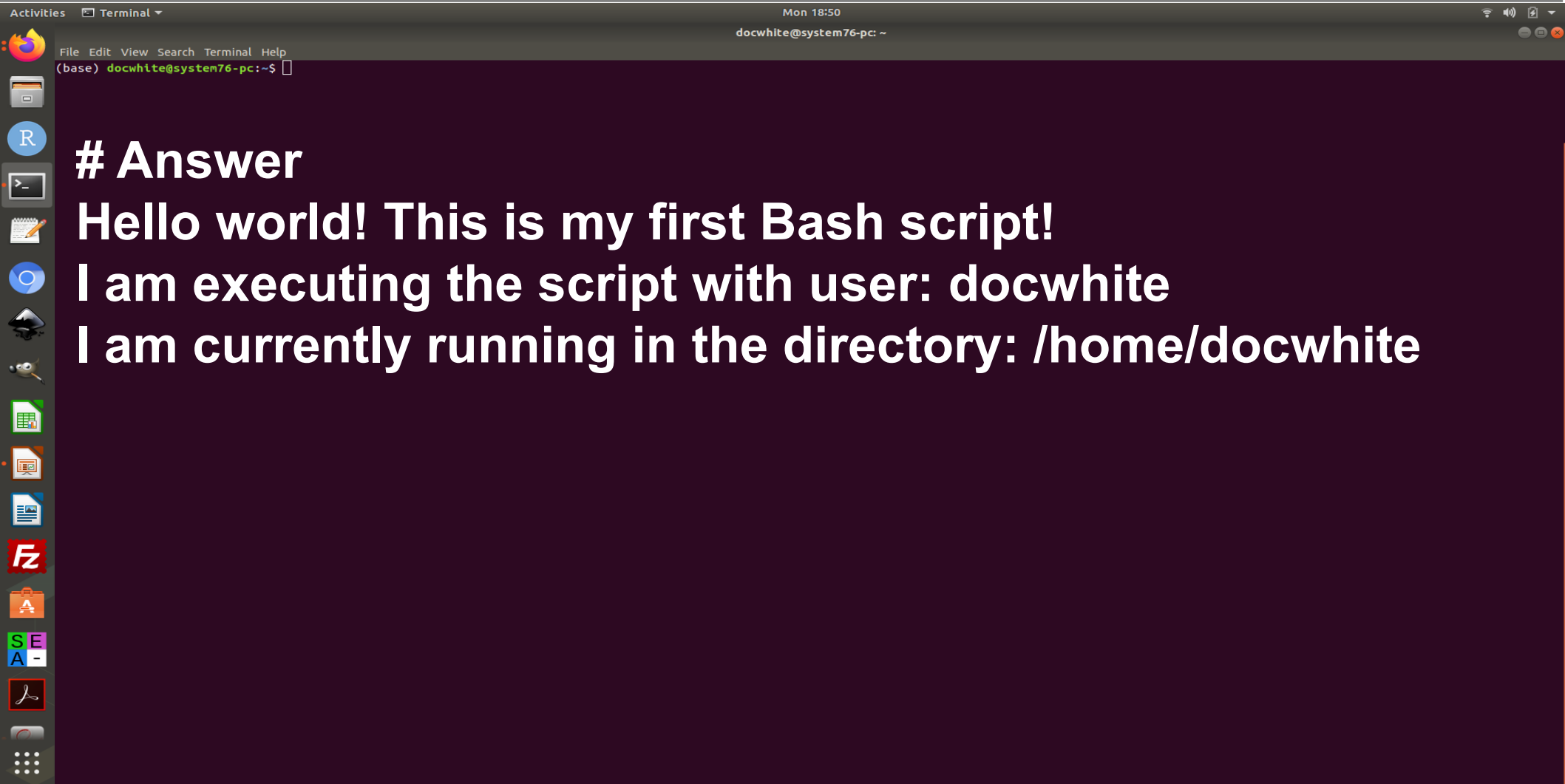
BASH examples



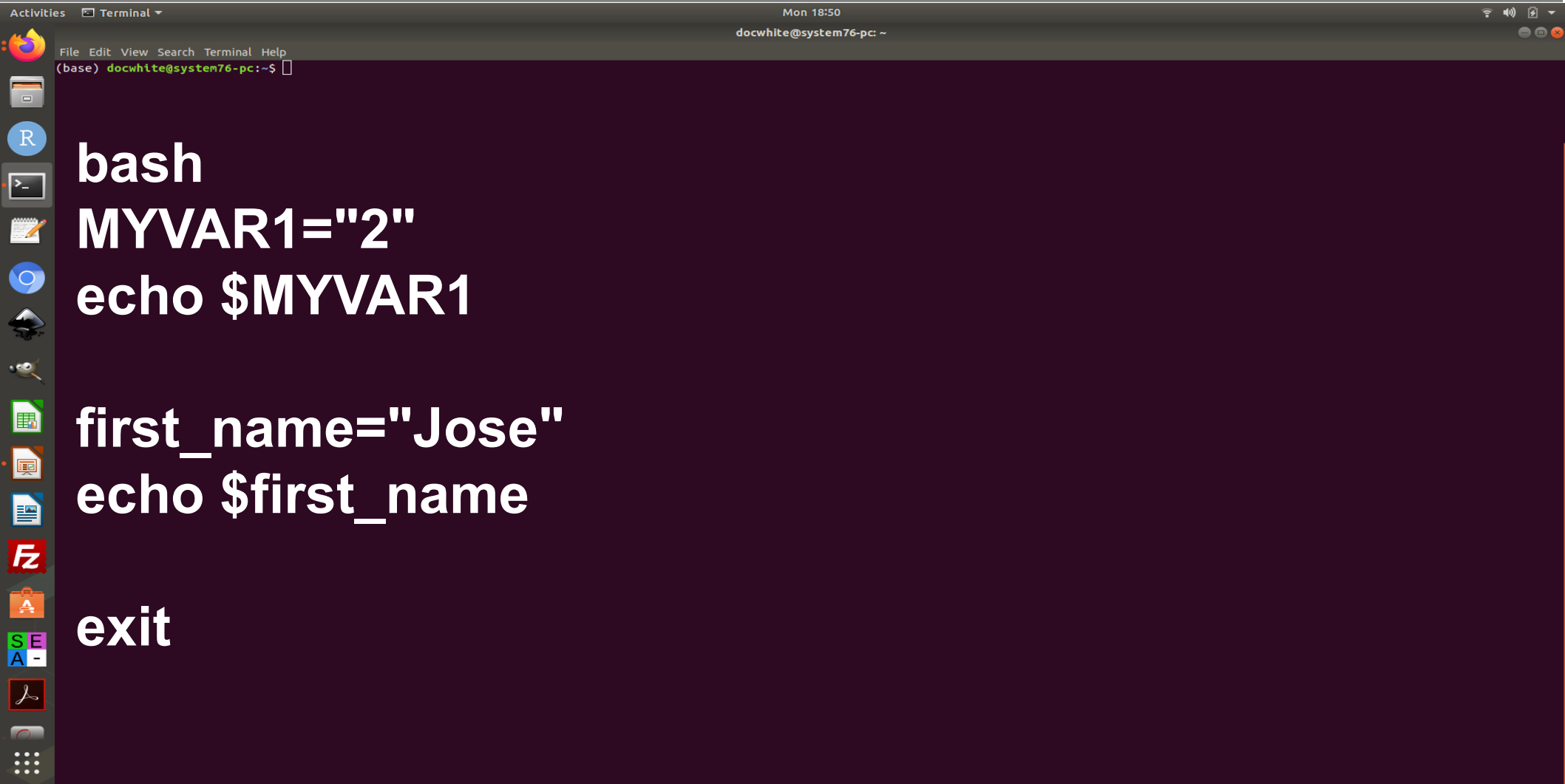
BASH examples



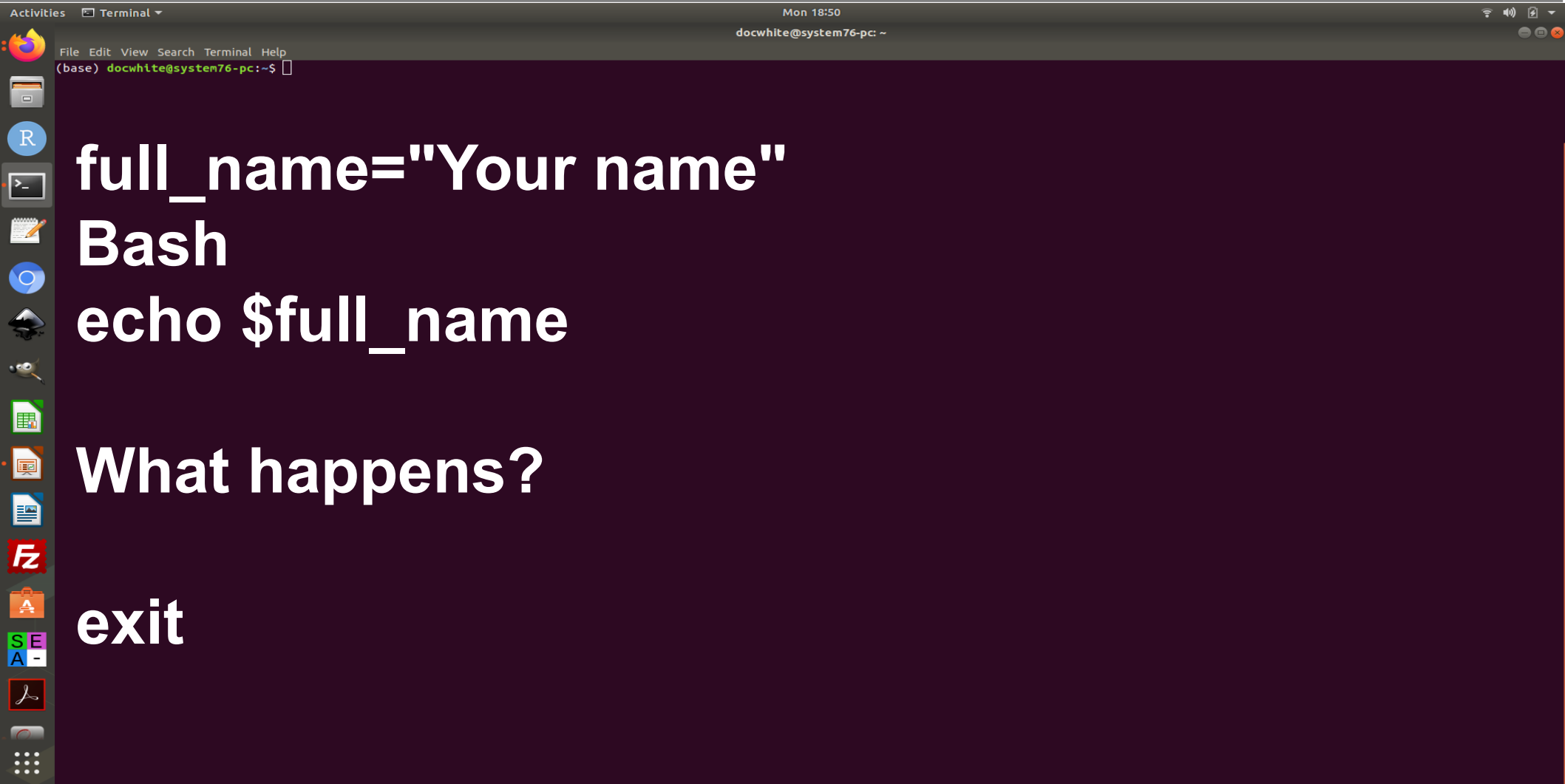
BASH examples



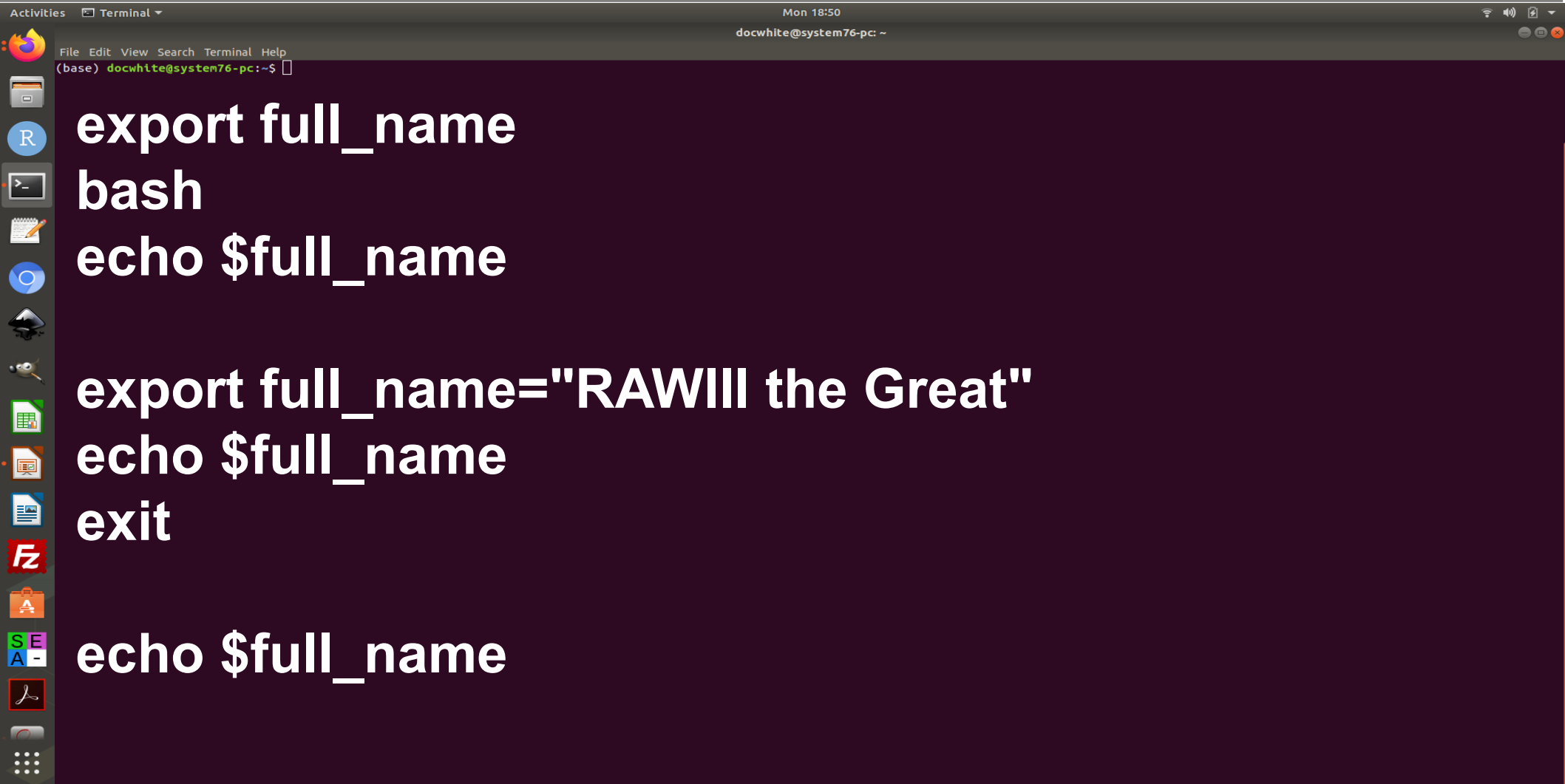
BASH variable creation



BASH variable creation



BASH variable creation



BASH Variables (pi_script.sh)

The best way to think of variables is as placeholders for values. They can be permanent (static) or transient (dynamic).

A variable is a temporary store for a piece of information. There are two actions we may perform for variables:

- Setting a value for a variable.
- Reading the value for a variable.

Variables may have their value set in a few different ways. The most common are to set the value directly and for its value to be set as the result of processing by a command or program.

What was the output?

```
1 #!/bin/bash
2
3 PI=3.14
4 VAR_A=10
5 VAR_B=$VAR_A
6 VAR_C=${VAR_B}
7
8 echo "Let's print 3 variables:"
9 echo $VAR_A
10 echo $VAR_B
11 echo $VAR_C
12
13 echo "We know this will break:"
14
15 #Since PIabc is not declared, it will be empty string
16 echo "0. The value of PI is $PIabc"
17
18 echo "And these will work:"
19 echo "1. The value of PI is $PI"
20 echo "2. The value of PI is ${PI}"
21 echo "3. The value of PI is" $PI
22
23 echo "And we can make a new string"
24
25 STR_A="Bob"
26 STR_B="Jane"
27 echo "${STR_A} + ${STR_B} equals Bob + Jane"
28 STR_C=${STR_A}" + "${STR_B}
29 echo "${STR_C} is the same as Bob + Jane too!"
30 echo "${STR_C} + ${PI}"
31
32 exit 0
```

BASH Variables (pi_script.sh)

What was the output?

Lets print 3 variables:

10

10

10

We know this will break:

0. The value of PI is

And these will work:

1. The value of PI is 3.14

2. The value of PI is 3.14

3. The value of PI is 3.14

And we can make a new string

Bob + Jane equals Bob + Jane

Bob + Jane is the same as Bob + Jane too!

Bob + Jane + 3.14

```
1 #!/bin/bash
2
3 PI=3.14
4 VAR_A=10
5 VAR_B=$VAR_A
6 VAR_C=${VAR_B}
7
8 echo "Let's print 3 variables:"
9 echo $VAR_A
10 echo $VAR_B
11 echo $VAR_C
12
13 echo "We know this will break:"
14
15 #Since PIabc is not declared, it will be empty string
16 echo "0. The value of PI is $PIabc"
17
18 echo "And these will work:"
19 echo "1. The value of PI is $PI"
20 echo "2. The value of PI is ${PI}"
21 echo "3. The value of PI is" $PI
22
23 echo "And we can make a new string"
24
25 STR_A="Bob"
26 STR_B="Jane"
27 echo "${STR_A} + ${STR_B} equals Bob + Jane"
28 STR_C=${STR_A}" + "${STR_B}
29 echo "${STR_C} is the same as Bob + Jane too!"
30 echo "${STR_C} + ${PI}"
31
32 exit 0
```

BASH Variables (pi_script.sh)

1st: We saw how we can use three variables, assign values to each of then, and print them.

2nd: We saw through a demonstration that the interpreter can break when concatenating strings (let's keep this in mind).

3rd: We printed out our PI variable and concatenated it to a string using echo.

Finally, we performed a few more types of concatenation, including a final version, which converts a numeric value and appends it to a string.

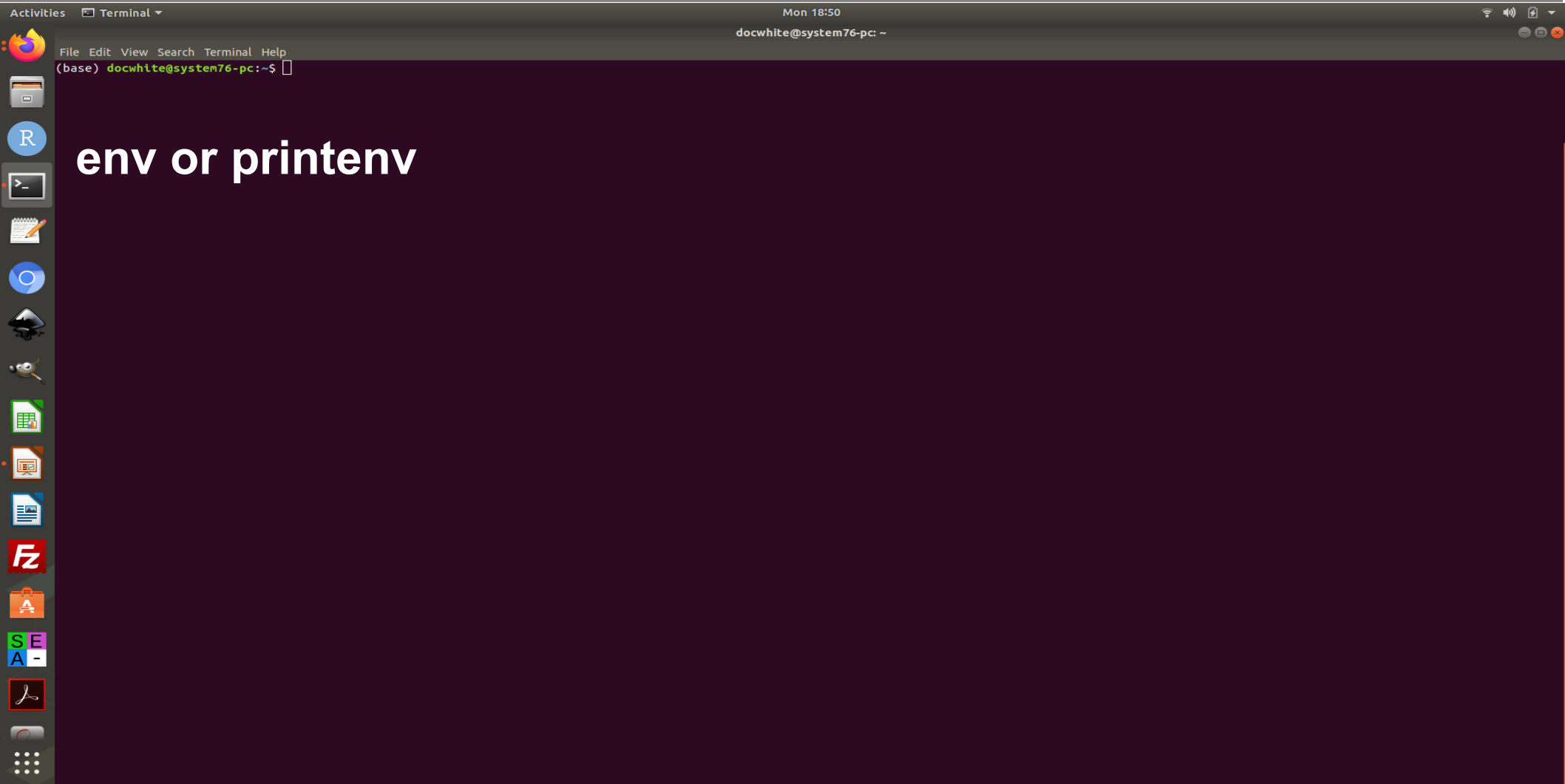
```
1 #!/bin/bash
2
3 PI=3.14
4 VAR_A=10
5 VAR_B=$VAR_A
6 VAR_C=${VAR_B}
7
8 echo "Let's print 3 variables:"
9 echo $VAR_A
10 echo $VAR_B
11 echo $VAR_C
12
13 echo "We know this will break:"
14
15 #Since PIabc is not declared, it will be empty string
16 echo "0. The value of PI is $PIabc"
17
18 echo "And these will work:"
19 echo "1. The value of PI is $PI"
20 echo "2. The value of PI is ${PI}"
21 echo "3. The value of PI is" $PI
22
23 echo "And we can make a new string"
24
25 STR_A="Bob"
26 STR_B="Jane"
27 echo "${STR_A} + ${STR_B} equals Bob + Jane"
28 STR_C=${STR_A}" + "${STR_B}
29 echo "${STR_C} is the same as Bob + Jane too!"
30 echo "${STR_C} + ${PI}"
31
32 exit 0
```

BASH Variables (Global vs. local)

Global variables or environment variables are available in all shells. The env or printenv commands can be used to display environment variables

Local variables are only available in the current shell. Using the set built-in command without any options will display a list of all variables (including environment variables) and functions. The output will be sorted according to the current locale and displayed in a reusable format.

BASH examples (Global variables)



BASH examples (Global variables)

BASH Variables (By content)

Apart from dividing variables in local and global variables, we can also divide them in categories according to the sort of content the variable contains.
In this respect, variables come in 4 types:

- String variables

- Integer variables

- Constant variables

- Array variables

BASH Variables (names)

Variables (legal):

myvar
MYVAR
Myvar
mYVAR
_myvar
my_var
myvar_
my012

Variables (Illegal):

1myvar
my-var
my.var
my:var

BASH special variables

$0 - The name of the Bash script.
$1 - $9 - The first 9 arguments to the Bash script.
$# - How many arguments were passed to the Bash script.
$@ - All the arguments supplied to the Bash script.
$? - The exit status of the most recently run process.
$$ - The process ID of the current script.
$USER - The username of the user running the script.
$HOSTNAME - The hostname of the machine the script is running on.
$SECONDS - The number of seconds since the script was started.
$RANDOM - Returns a different random number each time is it referred to.
$LINENO - Returns the current line number in the Bash script.

BASH Reserved words

! - Pipelines
[[ ]] - Conditional Constructs
{ } - Command Grouping
break - Looping Constructs
case - Conditional Constructs
continue - Looping Constructs
do - Looping Constructs
done - Looping Constructs
elif - Conditional Constructs
else - Conditional Constructs
esac - Conditional Constructs
fi - Conditional Constructs
for - Looping Constructs
function - Shell Functions
if - Conditional Constructs
in - Conditional Constructs
select - Conditional Constructs
then - Conditional Constructs
time - Pipelines
until - Looping Constructs
while - Looping Constructs

Similar from UNIX
& , | , > , < , ! , =
# , $, (,) , ; , {, } , [,] , \

https://www.gnu.org/software/bash/manual/html_node/Reserved-Word-Index.html

Quiz 7

- On canvas now

Bonus 7

- Write a bash script that states 150 kg at 178 cm is overweight?

ENJOY!