

BINF2111 - Introduction to Bioinformatics Computing

BASH 101 - Bash around and find out?



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

Lecture 7 - Tuesday Sep 12th, 2022

Learning Objectives

- 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 6

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

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.

```
war@war:~$ pwd
/home/war
war@war:~$ cd /usr/portage/app-shells/bash
war@war:~$ cd /usr/portage/app-shells/bash $ ls -al
total 136
drwxr-xr-x 3 portage portage 1824 Jul 25 18:06 .
drwxr-xr-x 33 portage portage 1824 Aug 7 22:39 ..
-rw-r--r-- 1 root root 35888 Jul 25 18:06 ChangeLog
-rw-r--r-- 1 root root 27882 Jul 25 18:06 Manifest
-rw-r--r-- 1 portage portage 4645 Mar 23 21:37 bash-3.1_p17.ebuild
-rw-r--r-- 1 portage portage 5977 Mar 23 21:37 bash-3.2_p39.ebuild
-rw-r--r-- 1 portage portage 6151 Apr 5 14:37 bash-3.2_p48-r1.ebuild
-rw-r--r-- 1 portage portage 5988 Mar 23 21:37 bash-3.2_p48.ebuild
-rw-r--r-- 1 portage portage 5643 Apr 5 14:37 bash-4.0_p18-r1.ebuild
-rw-r--r-- 1 portage portage 6238 Apr 5 14:37 bash-4.0_p18.ebuild
-rw-r--r-- 1 portage portage 5648 Apr 14 05:52 bash-4.0_p17-r1.ebuild
-rw-r--r-- 1 portage portage 5532 Apr 8 18:21 bash-4.0_p17.ebuild
-rw-r--r-- 1 portage portage 5668 May 30 03:35 bash-4.0_p24.ebuild
-rw-r--r-- 1 root root 5668 Jul 25 09:43 bash-4.0_p28.ebuild
drwxr-xr-x 2 portage portage 2848 May 30 03:35 files
-rw-r--r-- 1 portage portage 468 Feb 9 04:35 metadata.xml
war@war:~$ cd /usr/portage/app-shells/bash $ cat metadata.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "http://www.gentoo.org/dtd/metadata.dtd">
<pkgmetadata>
  <herd>base-system</herd>
  <use>
    <flag name="bashlogger">Log ALL commands typed into bash; should ONLY be
      used in restricted environments such as honeypots</flag>
    <flag name="net">Enable /dev/tcp/host/port redirection</flag>
    <flag name="plugins">Add support for loading builtins at runtime via
      'enable'</flag>
  </use>
</pkgmetadata>
war@war:~$ cd /usr/portage/app-shells/bash $ sudo /etc/init.d/bluetooth status
Password:
* status: started
war@war:~$ cd /usr/portage/app-shells/bash $ ping -q -c1 en.wikipedia.org
PING rr.esaws.wikiimedia.org (91.196.174.2) 56(84) bytes of data:

--- rr.esaws.wikiimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 2ms
rtt min/avg/max/mdev = 49.028/49.028/49.028/0.000 ms
war@war:~$ cd /usr/portage/app-shells/bash $ grep -l /dev/sda /etc/fstab | cut --fields=3
/dev/sda1 /boot
/dev/sda2 none
/dev/sda3 /
war@war:~$ cd /usr/portage/app-shells/bash $ date
Sat Aug 8 02:42:24 HST 2009
war@war:~$ cd /usr/portage/app-shells/bash $ lsmod
Module Size Used by
rmdisk_wlan 23424 0
rmdisk_host 8696 1 rmdisk_wlan
cdc_ether 5672 1 rmdisk_host
usbnet 18688 3 rmdisk_wlan,rmdisk_host,cdc_ether
parport_pc 38424 0
fglrx 2388128 20
parport 39648 1 parport_pc
l7CO_wdt 12272 0
l2c_i801 9398 0
war@war:~$ cd /usr/portage/app-shells/bash $
```

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	Globbering 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
csch	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.

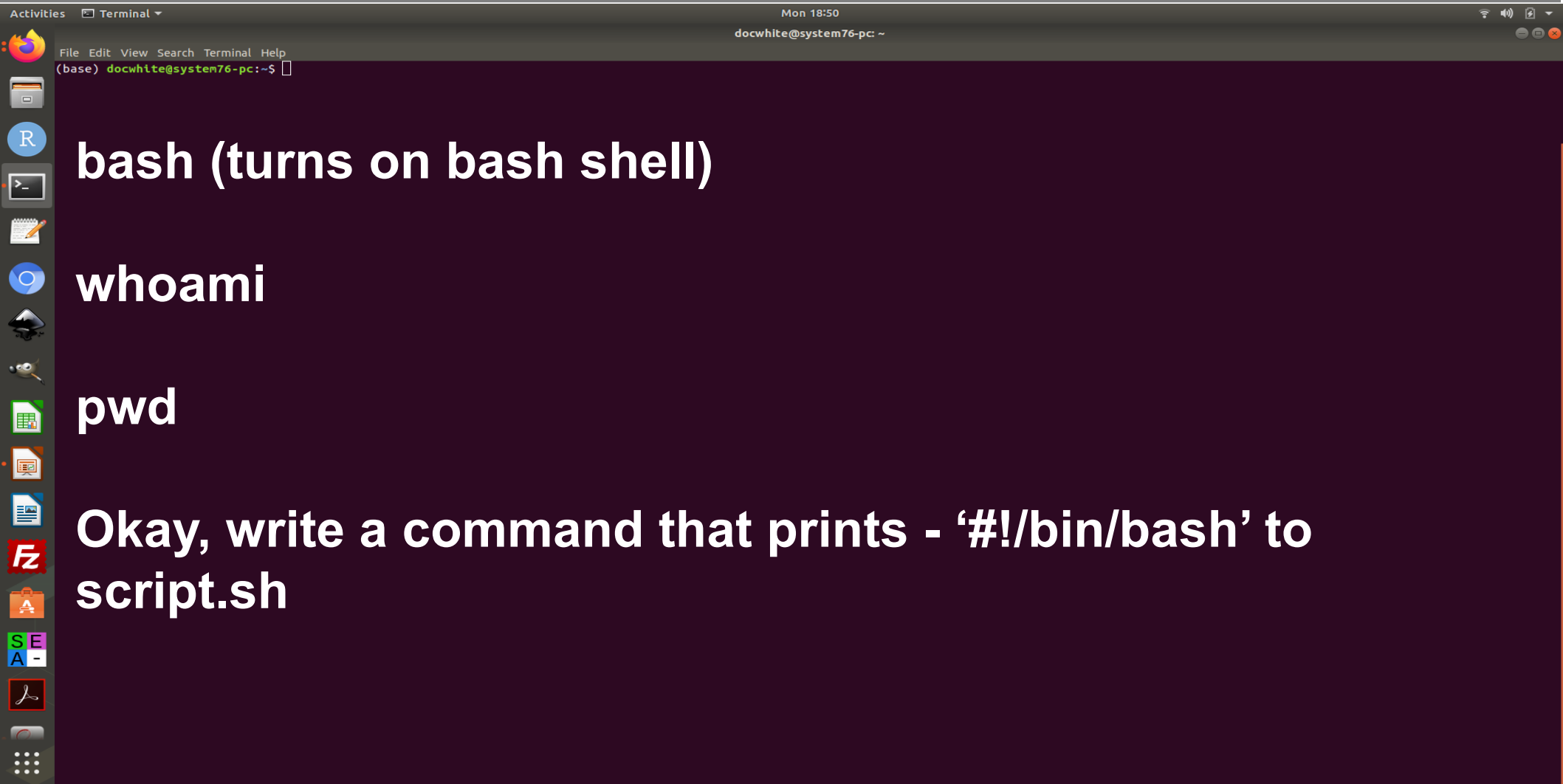
Bash was also the default shell in all versions of Apple macOS prior to the 2019 release of macOS Catalina, which changed the default shell to zsh.

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$ pwd
/home/ars
bash$ cd /usr/portage/app-shells/bash
bash$ ls -al
total 136
drwxr-xr-x 3 portage portage 1024 Jul 25 10:06 .
drwxr-xr-x 33 portage portage 1024 Aug 7 22:39 ..
-rw-r--r-- 1 root root 35880 Jul 25 10:06 ChangeLog
-rw-r--r-- 1 root root 27082 Jul 25 10:06 Manifest
-rw-r--r-- 1 portage portage 4645 Mar 23 21:37 bash-3.1_p17.ebuild
-rw-r--r-- 1 portage portage 5977 Mar 23 21:37 bash-3.2_p39.ebuild
-rw-r--r-- 1 portage portage 6151 Apr 5 14:37 bash-3.2_p48-r1.ebuild
-rw-r--r-- 1 portage portage 5988 Mar 23 21:37 bash-3.2_p48.ebuild
-rw-r--r-- 1 portage portage 5643 Apr 5 14:37 bash-4.0_p10-r1.ebuild
-rw-r--r-- 1 portage portage 6238 Apr 5 14:37 bash-4.0_p10.ebuild
-rw-r--r-- 1 portage portage 5648 Apr 14 05:52 bash-4.0_p17-r1.ebuild
-rw-r--r-- 1 portage portage 5532 Apr 8 10:21 bash-4.0_p17.ebuild
-rw-r--r-- 1 portage portage 5668 May 30 03:35 bash-4.0_p24.ebuild
-rw-r--r-- 1 root root 5568 Jul 25 09:43 bash-4.0_p28.ebuild
drwxr-xr-x 2 portage portage 2048 May 30 03:35 files
-rw-r--r-- 1 portage portage 460 Feb 9 04:35 metadata.xml
bash$ cat metadata.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "http://www.gentoo.org/dtd/metadata.dtd">
<pkgmetadata>
  <hardbase-system/>
  <use>
    <flag name="bashlogger">Log ALL commands typed into bash; should ONLY be
    used in restricted environments such as honeypots</flag>
    <flag name="net">Enable /dev/tcp/host/port redirection</flag>
    <flag name="plugins">Add support for loading builtins at runtime via
    'enable' </flag>
  </use>
</pkgmetadata>
bash$ sudo /etc/init.d/bluetooth status
Password:
* status: started
bash$ ping -q -c 1 en.wikipedia.org
PING rr.esams.wikiimedia.org (91.190.174.2) 56(84) bytes of data:

--- rr.esams.wikiimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 2ms
rtt min/avg/max/mdev = 49.020/49.020/49.020/0.000 ms
bash$ grep -l /dev/sda /etc/fstab | cut -f1 -d=
/dev/sda1 /boot
/dev/sda2 none
/dev/sda3 /
bash$ date
Sat Aug 8 02:42:24 HST 2009
bash$ lsmod
Module Size Used by
rmdis_wlan 23424 0
rmdis_host 8696 1 rmdis_wlan
cdc_ether 5672 1 rmdis_host
usbnet 10688 3 rmdis_wlan,rmdis_host,cdc_ether
perport_pc 30424 0
fglrx 2388128 20
perport 39648 1 perport_pc
l7CO_wdt 12272 0
l2c_i801 9380 0
bash$
```

BASH examples



bash (turns on bash shell)

whoami

pwd

Okay, write a command that prints - '#!/bin/bash' to script.sh

BASH examples

```
printf '#!/bin/bash\n' >script.sh
```

or

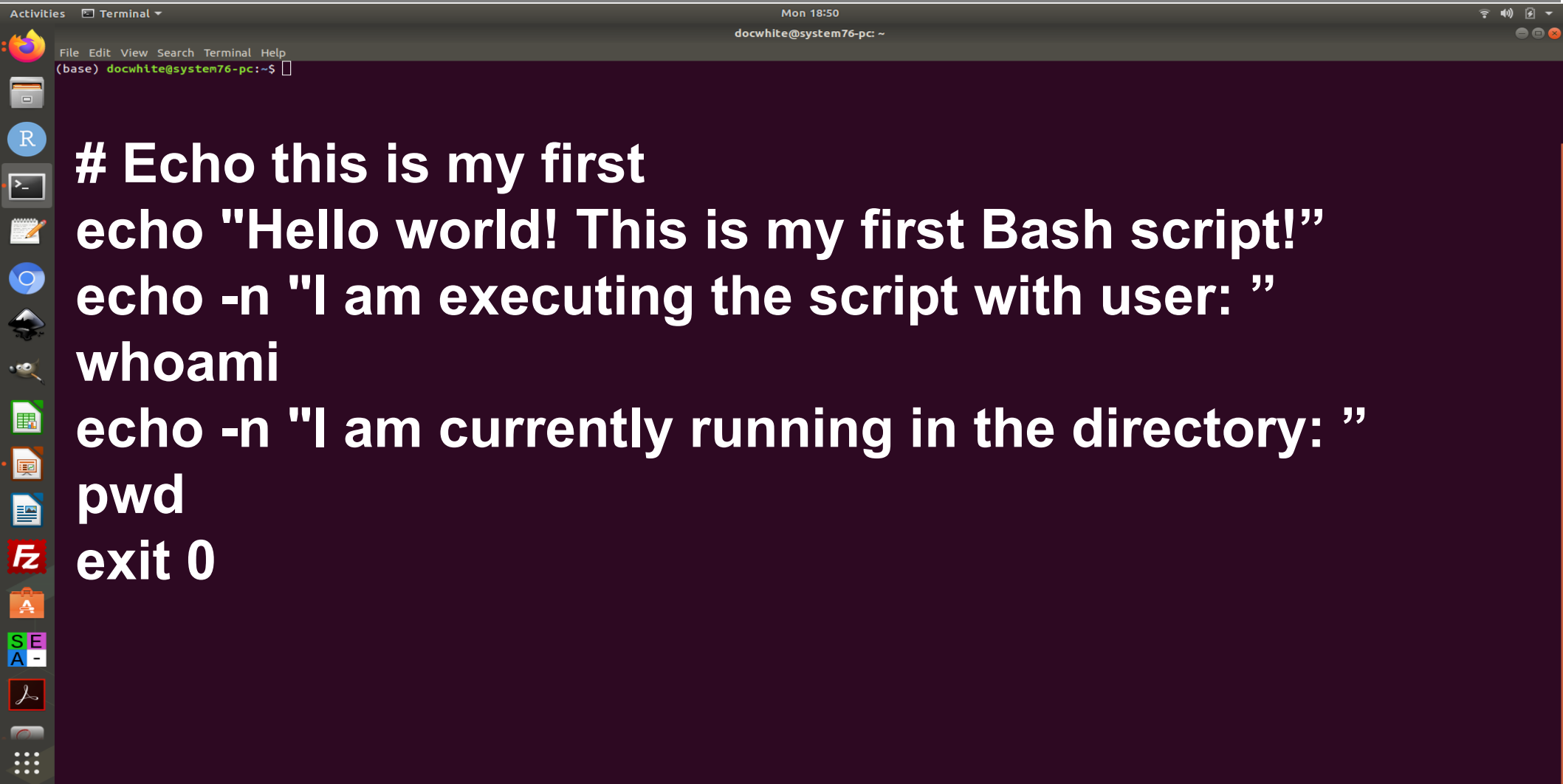
BASH examples

```
printf '#!/bin/bash\n' >script.sh
```

or

echo '#!/bin/bash' >script.sh

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: "

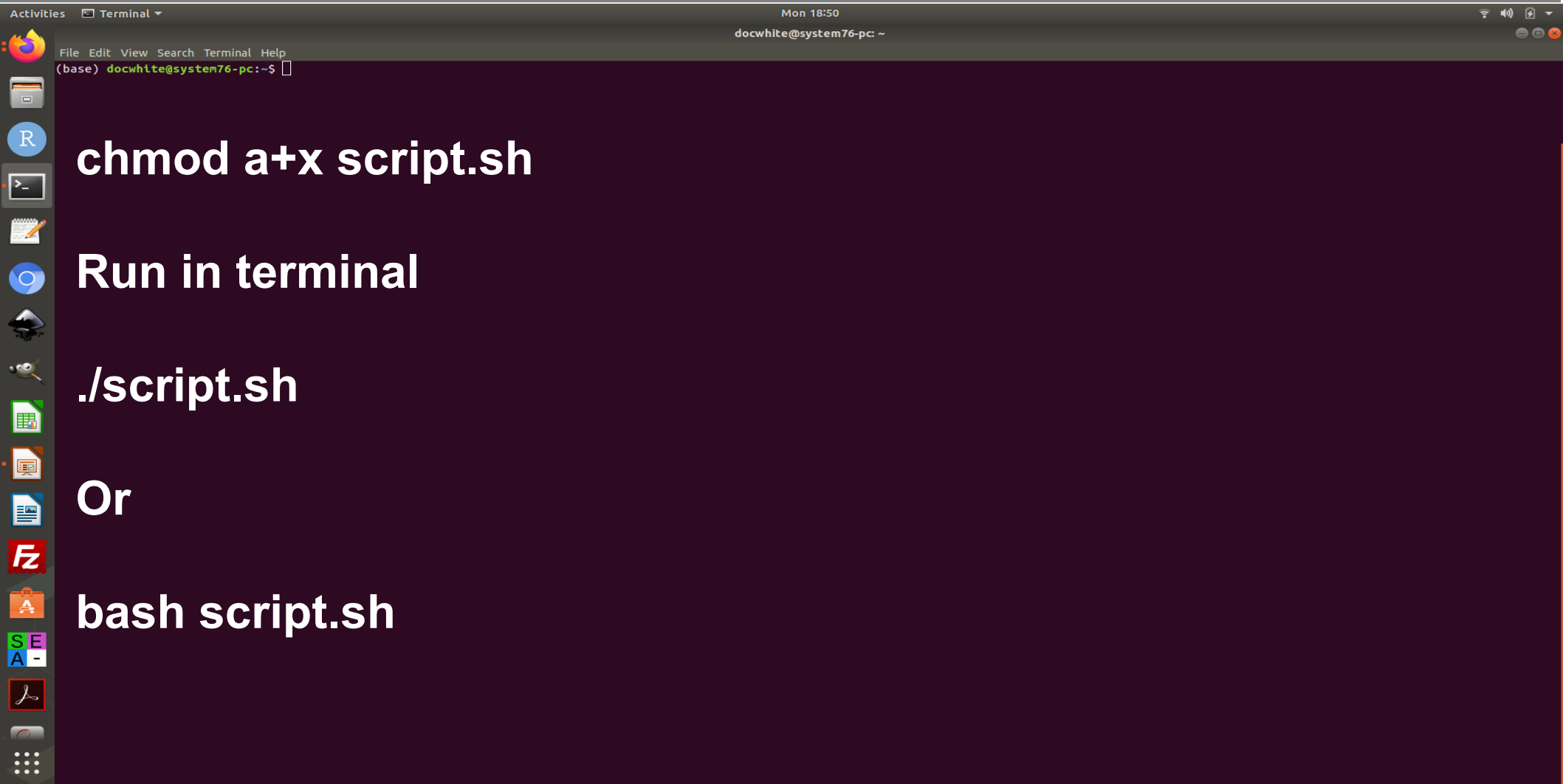
pwd

exit 0

Activities Terminal Mon 18:50

```
(base) docwhite@system76-pc:~$  
# Echo this is my first  
printf '#!/bin/bash  
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\n' >>script.sh
```

BASH examples (compiling)



chmod a+x script.sh

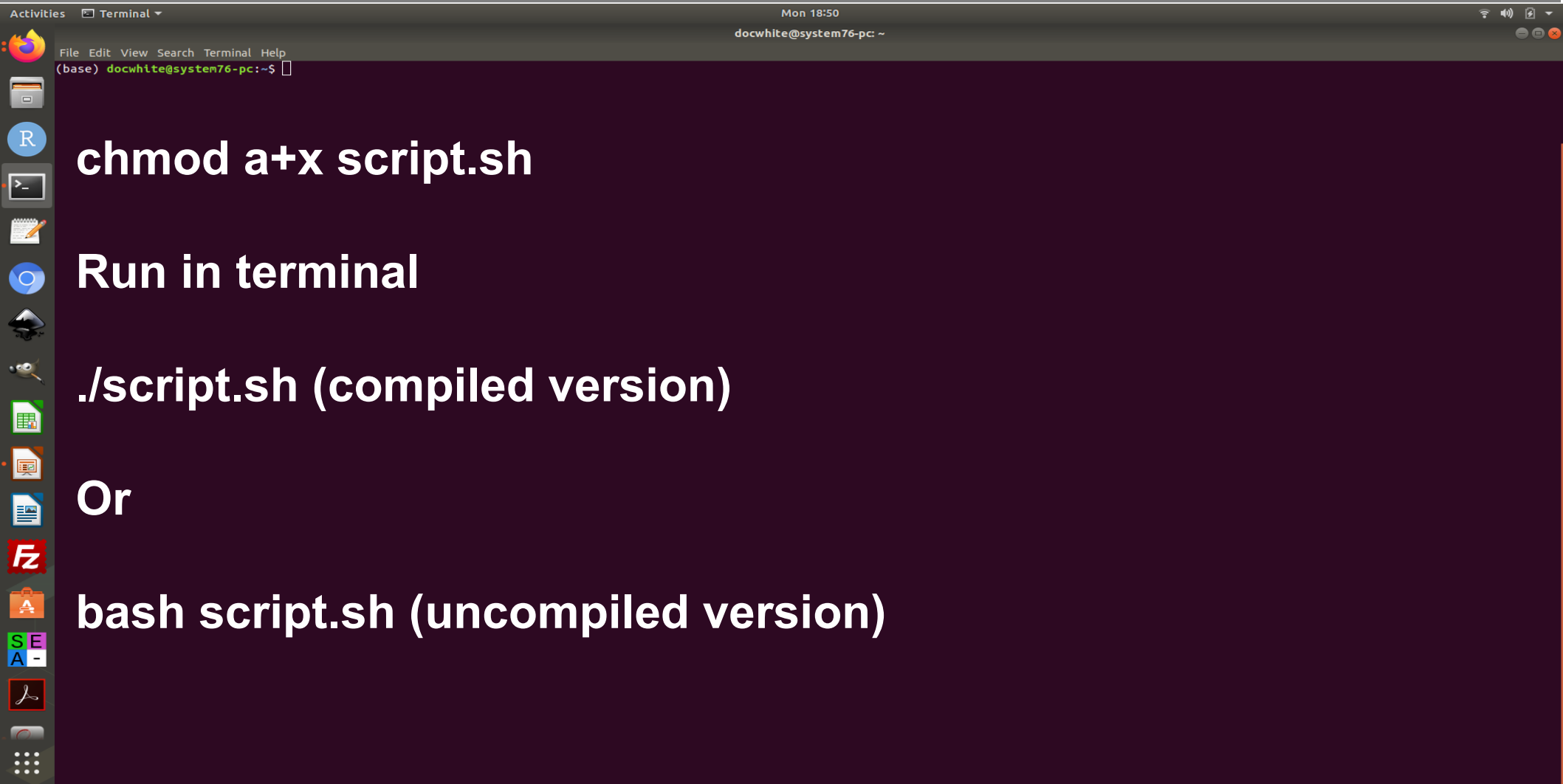
Run in terminal

./script.sh

Or

bash script.sh

BASH examples (compiling)



```
chmod a+x script.sh
```

Run in terminal

```
./script.sh (compiled version)
```

Or

```
bash script.sh (uncompiled version)
```

BASH examples

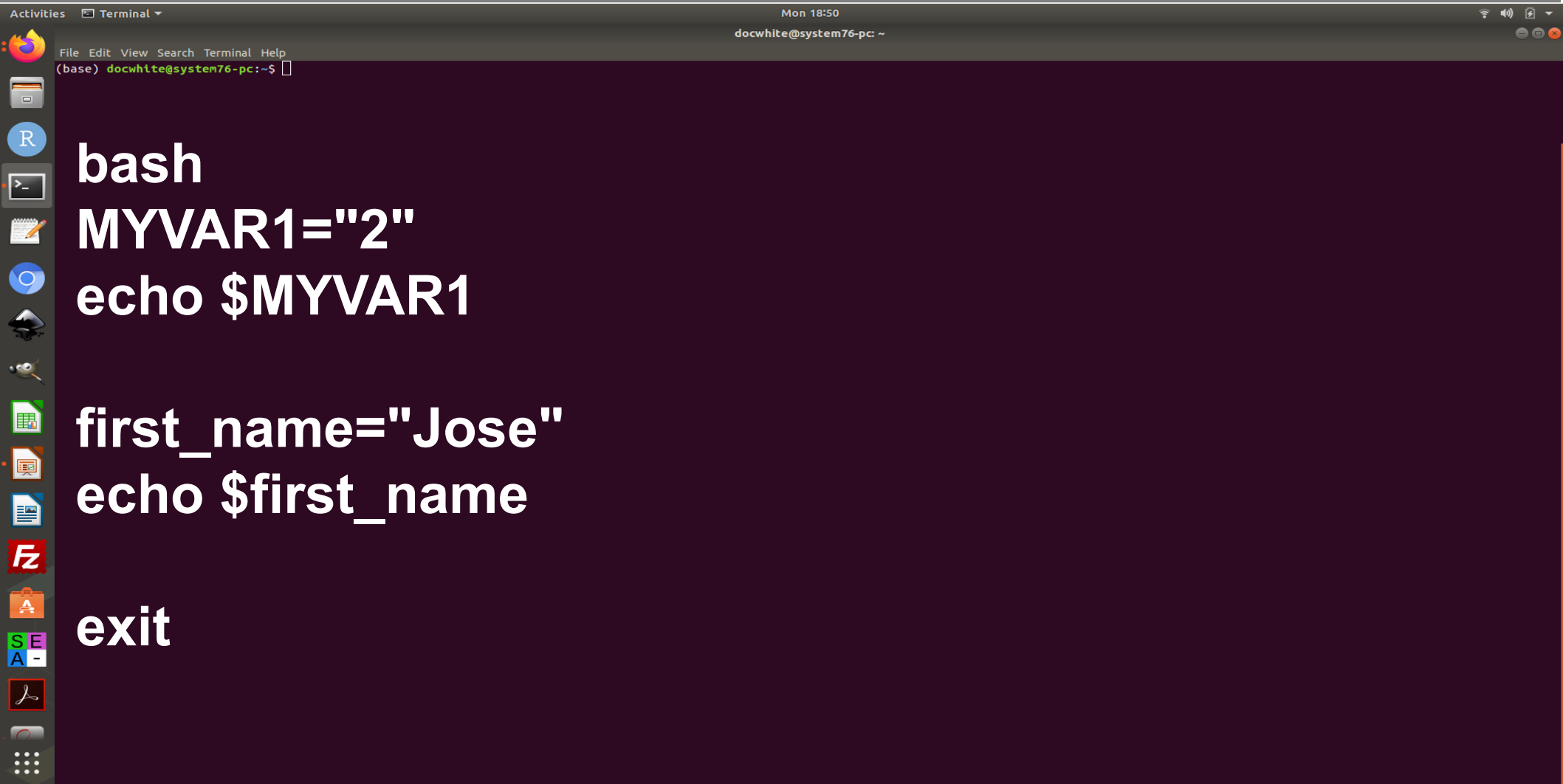
Answer

Hello world! This is my first Bash script!

I am executing the script with user: docwhite

I am currently running in the directory: /home/docwhite

BASH variable creation



```
bash
```

```
MYVAR1="2"
```

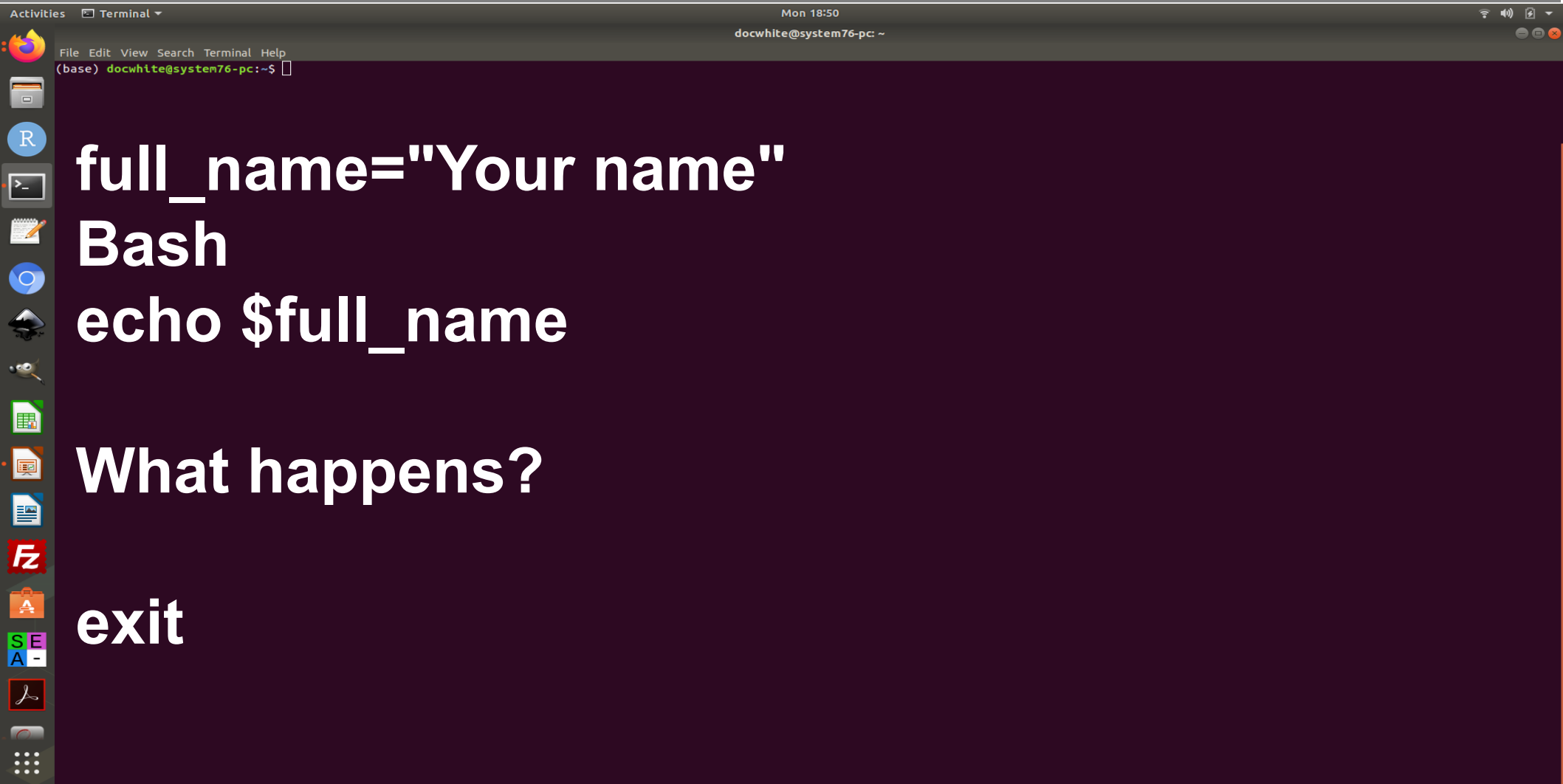
```
echo $MYVAR1
```

```
first_name="Jose"
```

```
echo $first_name
```

```
exit
```

BASH variable creation



```
full_name="Your name"
```

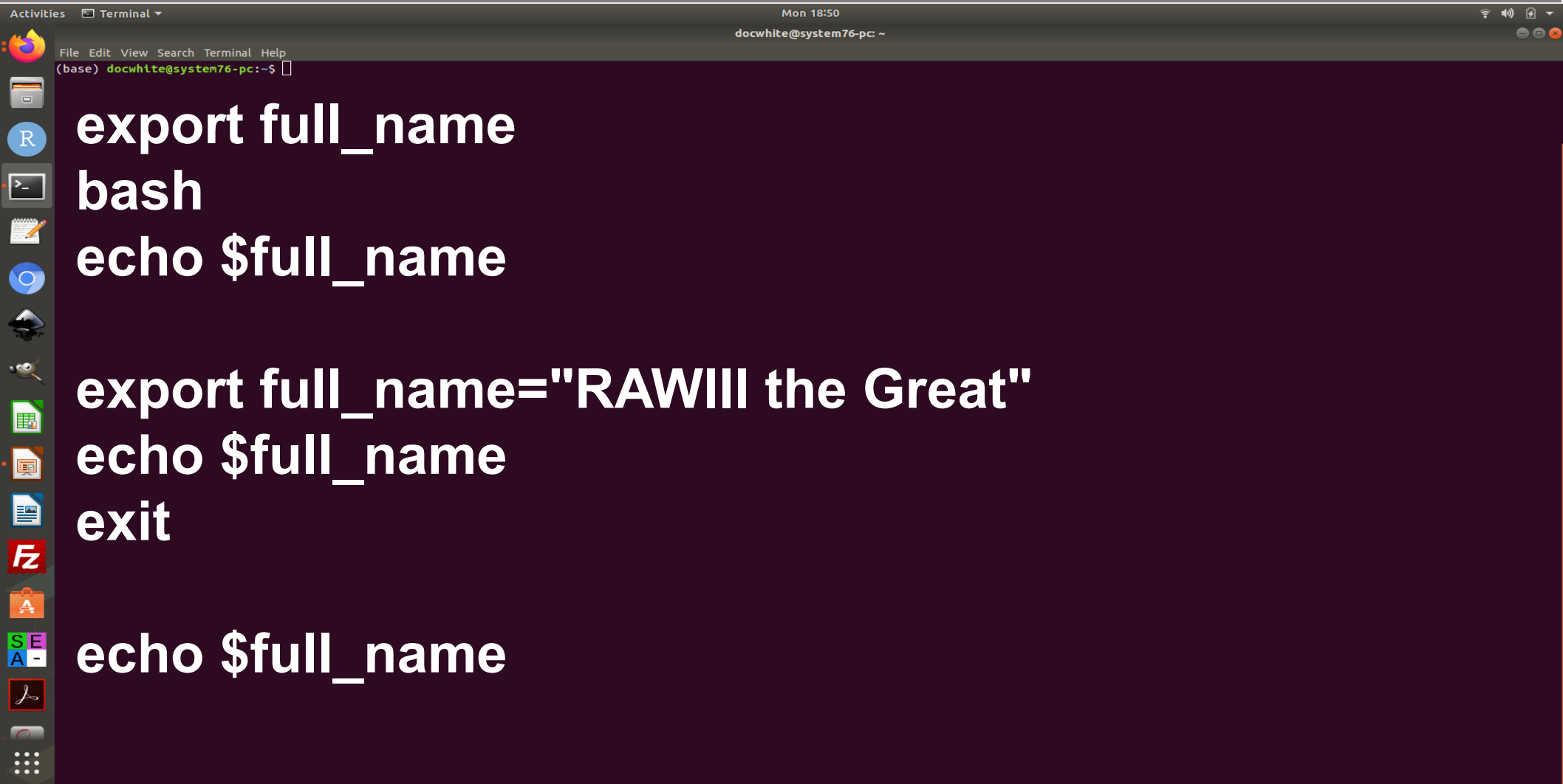
```
Bash
```

```
echo $full_name
```

```
What happens?
```

```
exit
```

BASH variable creation

A screenshot of a Linux terminal window. The title bar at the top shows 'Activities', 'Terminal', and the time 'Mon 18:50'. The terminal text shows the user 'docwhite' at 'system76-pc' in a '~' directory. The prompt is '(base) docwhite@system76-pc:~\$'. The commands entered are: 'export full_name', 'bash', 'echo \$full_name', 'export full_name="RAWIII the Great"', 'echo \$full_name', 'exit', and 'echo \$full_name'.

```
Mon 18:50
docwhite@system76-pc: ~
(base) docwhite@system76-pc:~$
export full_name
bash
echo $full_name

export full_name="RAWIII the Great"
echo $full_name
exit
echo $full_name
```


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)

st

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

nd

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

rd

3 : 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)

env or printenv

BASH examples (Global variables)

Activities Terminal Mon 18:50 docwhite@system76-pc: ~

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

env or printenv

CLUTTER_IM_MODULE=xim

CONDA_SHLVL=1

NVM_DIR=/home/docwhite/.nvm

LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01
:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=3
7;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.l
z4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=

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 it is 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?