BINF2111 – Introduction to Bioinformatics Computing

MID-TERM REVIEW

Richard Allen White III, PhD
RAW Lab

Lecture 14 – Thursday Oct 5th, 2023

Learning Objectives

- Review!

- Review!!

- Review!!!

- Review!!!!

- Review!!!!!

- Quiz 14

Set-up to the exam

Lab exam (50 pts)

- Fill in commands
- Scripts (BASH)

Lecture exam (50 pts)

- Multiple choice
- True/False
- 25 questions

Topics covered in the exam

Command line (UNIX)
Text wrangling and manipulation (awk/grep/sed/cut/tr/perl)
Regular expressions (AWK/grep/sed/perl)
Bash scripting (loops, variables, functions, conditionals)
Github
Markdown in Github
Slurm
Text editors
Cloud/Cluster

Bioawk and conda

```
conda create -n bioawk
conda activate bioawk
conda install -c bioconda bioawk

bioawk -c fastx '{ print $name, length($seq) }' example2.fasta
bioawk -c fastx '{ print $name, gc($seq) }’ example2.fasta
bioawk -c fastx '{ print ">"$name;print revcomp($seq) }'
example2.fasta
```

https://bioinformaticsworkbook.org/Appendix/Unix/bioawk-basics.html#gsc.tab=0

Types of files

FASTA AMINO ACID = .faa
FASTA NUCLEOTIDE + QUALITY = .fastq or .fq
FASTA NUCLEOTIDE = .fna

Also, Dr. White and his lab studies VIRUSES.

printf – syntax anatomy UNIX

printf [-v var] format [arguments]

--help display this help and exit
--version output version information and exit

\xHH byte with hexadecimal value HH (1 to 2 digits)
\uHHHH Unicode (ISO/IEC 10646) character with hex value HHHH (4 digits)
\UHHHHHHHH
Unicode character with hex value HHHHHHHH (8 digits)
%% a single %
%b ARGUMENT as a string with '\' escapes interpreted, except that octal escapes are of the form \0 or \0NNN

\" double quote
\NNN character with octal value NNN (1 to 3 digits)
\\ backslash
\a alert (BEL)
\b backspace
\c produce no further output
\f form feed
\n new line
\r carriage return
\t horizontal tab
\v vertical tab

echo like printf - syntax anatomy UNIX

echo format [arguments]

echo -n -e "This is a line without a newline.\n"

output
This is a line without a newline

echo -n -e "This\n \n \nThis\n" >out.txt

output
This

This

echo -n -e ‘#!/bin/bash’ >script.sh

output
#!/bin/bash

echo -n -e ‘#!/bin/bash\n \npython.py input.tsv output.csv’ >script.sh

output
#!/bin/bash

python.py input.tsv output.csv

Sed – S swiss army knife (regular expressions)

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 – S swiss army knife (regular expressions)

More
DaVid,abdul,xi,BilL
Mary,dAvid,Bill,aBdul

Examples
(convert all to lower case): sed 's/[A-Z]/\L&/g' file.csv >file_lcas.csv

(convert all to upper case): sed 's/[a-z]/\U&/g' file.csv >file_ucas.csv

Sed – Special characters (regular expressions)

more file.csv
DaVid,a$dul,xi,BilL
M*ry,d#vid,Bill,a%dul

Examples

sed 's/[$]/b/g' file.csv >file_fixed.csv
sed 's/[%]/b/g' file.csv >file.csv or sed 's/%/b/g' file.csv >file_fixed.csv
sed 's/[*]/a/g' file.csv >file.csv or sed 's/*/a/g' file.csv >file_fixed.csv
sed 's/[#]/a/g' file.csv >file.csv or sed 's/#/a/g' file.csv >file_fixed.csv

Delete empty lines

- Delete all the ‘all white space/empty lines’

→ grep: grep -v -e '^[[:space:]]*$' file

→ grep -v -P '^\s*$' file

→ awk: awk 'NF > 0' file

→ awk: awk 'NF' file

→ sed '/^[[:space:]]*$/d' file

→ sed '/^ *$/d' file

Perl like Grep, Sed and Awk functions

```
# check perl --help
# -e means single line expression (a raw regular expression is in fact
an executable expression in perl)
# -n means execute on each line
# -p means execute on each line and print the result
# -F... means split the source text using the following pattern ...
# -a is part of -F, and splits the source text into @F[...]
# -l means print everything with a separator, by default newlines
```

Perl like Grep, Sed and Awk functions

check perl --help
-e means single line expression (a raw regular expression is in fact an executable expression in perl)
-n means execute on each line
-p means execute on each line and print the result
-F... means split the source text using the following pattern ...
-a is part of -F, and splits the source text into @F[...]
-l means print everything with a separator, by default newlines

Grep like:

```
perl -ne 'print if /chr1_geneA/' example2.fasta | more
perl -ne 'print if /chr1_geneB/' example2.fasta | more
```

Perl like Grep, Sed and Awk functions

check perl --help
-e means single line expression (a raw regular expression is in fact an executable expression in perl)
-n means execute on each line
-p means execute on each line and print the result
-F... means split the source text using the following pattern ...
-a is part of -F, and splits the source text into @F[...]
-l means print everything with a separator, by default newlines

sed like:
perl -pe 's/chr1/chr2/' example2.fasta | more (without replacement)
perl -i -pe 's/chr1/chr2/' example2.fasta | more (with replacement)

Perl like Grep, Sed and Awk functions

check perl --help
-e means single line expression (a raw regular expression is in fact
an executable expression in perl)
-n means execute on each line
-p means execute on each line and print the result
-F... means split the source text using the following pattern ...
-a is part of -F, and splits the source text into @F[...]
-l means print everything with a separator, by default newlines

awk like:
cat /etc/passwd | awk -F: '{ print $1 }'
cat /etc/passwd | perl -F: -lane 'print @F[0]'

cut – syntax anatomy UNIX's scissors

cut [options] file.txt

-d (--delimiter) "," set field delimiter (default tab)
-f (--fields=LIST) Select by specifying a field
    -f 2 select a field to cut (left is 1)
    -f 2-8,12 select multiple fields to cut

For example:
Grab columns 1,2
cut -f1,2 file.tsv | more

PATHS - how to edit?

export PATH=$PATH:/new/path

Or

Edit your .bashrc file

gedit .bashrc or other text editor

Grep and AWK regular expressions

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

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

Does awk ‘/[Ss]teven\tJose’ file.tsv | wc -l work?
awk ‘/[Ss]teven\tJose/’ file.tsv | wc -l

Perl regular expression (sed style)

My input is:
more file.tsv
bill rod david
Xi abdul larry

My output is:
more file.csv
bill,rod,david
Xi,abdul,larry

perl -pi -e 's/\t/,/' file.tsv

T or F

Perl regular expression (sed style)

My input is:
more file.tsv
bill rod david
Xi abdul larry

My output is:
more file.csv
bill,rod david
Xi,abdul larry

perl -pi -e 's/\t/,/' file.tsv

T or F

perl -pi -e 's/\t/,/g' file.tsv

Line number counting (range)

more file.csv
David,abdul,xi,Bill
Mary,david,bill,abdul
Wang,AbDul,xi,Bavid
BIll,maRy,steve,Su
B*ll,A#dul,Xi,zOra

cat -n file.csv | sed -n ‘1,3p’ (prints line 1 through 3)
cat -n file.csv | head -3 | tail +1 (prints line 1 through 3)
cat -n file.csv | awk 'NR>=1 && NR<=3' (prints line 1 through 3)

Line number counting (specific lines)

more file.csv
David,abdul,xi,Bill
Mary,david,bill,abdul
Wang,AbDul,xi,Bavid
BIll,maRy,steve,Su
B*ll,A#dul,Xi,zOra

cat -n file.csv | sed -n ‘1p;3p’ (prints line 1 and 3 ONLY)

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
```

Our favorite question is?

Converting TSV to CSV

&

Converting CSV to TSV

STUDY THIS!

Converting CSV to TSV

With this file (file.csv):
Rat,steven,bear,Xi
Olf,thor,flower,Ton
I WOULD KNOW THREE – Hint, hint!

Command 1: sed ‘s/,/\t/g’ file.csv >file.tsv
Command 2: cat file.csv | tr -s ',', '\t' >file.tsv
Command 3: awk ‘{gsub(",", "\t"); print}’ file.csv > file.tsv
Command 4: perl -pi -e 's/,/\t/g' file.csv > file.tsv
Command 5: cat file.csv | awk -F ',' '{$1=$1}1' >file.tsv

Converting CSV to TSV

- Write a bash script that converts this into a csv file (**specific files** and all files)?

```
1 #!/bin/bash/
2
3 input=$1
4
5 sed 's/,/\t/g' $input
6
7 function print_to_terminal(){
8     echo "Your comma-separated file has been converted to tab-delim file" >$(tty)
9     echo -n "Wise choice!" >$(tty)
10 }
11
12 output=$(print_to_terminal)
13
```

Converting CSV to TSV

- Write a bash script that converts this into a csv file (specific files and **all files**)?

```
#!/bin/bash/

for i in *csv;
do
    sed 's/,/\t/g' "$i" >$(basename "$i" .csv).tsv
done

function print_to_terminal(){
    echo "Your comma-separated file has been converted to tab-delim file" >$(tty)
    echo -n "Wise choice!" >$(tty)
}

output=$(print_to_terminal)
```

Next favorite question is?

Converting DNA to Amino Acids

STUDY THIS!

Amino Acid conversation

Write a bash script that counts the number of ATG (starts), Serine (S), Arginine (R), and TAA, TAG, TGA (stops) from the example2.fasta file then converts them into amino acid M, S, R, and * for TAA, TAG, TGA for stop codon.

Remember that ATG encodes for methonine so they only count as start from the beginning of the sequence or the end for the stops. Serine and Arginine can be throughout the sequence.

HOW WOULD YOU DO THIS?

Amino Acid conversation

```bash
#!/bin/bash

input=$1

#Count start codons
count_starts(){
    grep "^ATG" $input | wc -l
}

number_starts=$(count_starts)

#Count stop codons
count_stops(){
    egrep "TAA$|TAG$|TGA$" $input | wc -l
}

number_stops=$(count_stops)

#Count codons for arginine
count_arg(){
    egrep "CGT|CGC|CGA|CGG|AGA|AGG" $input | wc -l
}

number_arg=$(count_arg)

#Count codons for serine
count_ser(){
    egrep "AGT|AGC|TCT|TCC|TCA|TCG" $input | wc -l
}

number_ser=$(count_ser)

#final table
echo -n "Number of starts: "
echo $number_starts
echo -n "Number of stops: "
echo $number_stops
echo -n "Number of Arginine: "
echo $number_arg
echo -n "Number of Serine: "
echo $number_ser
```

```bash

#Count codons for arginine
count_arg(){
    egrep "CGT|CGC|CGA|CGG|AGA|AGG" $input | wc -l
}

number_arg=$(count_arg)

#Count codons for serine
count_ser(){
    egrep "AGT|AGC|TCT|TCC|TCA|TCG" $input | wc -l
}

number_ser=$(count_ser)

#final table
echo -n "Number of starts: "
echo $number_starts
echo -n "Number of stops: "
echo $number_stops
echo -n "Number of Arginine: "
echo $number_arg
echo -n "Number of Serine: "
echo $number_ser

#Convert codons
convert_cod(){
    sed 's/^ATG/M/g' $input |
    sed 's/TAA$/*/g' |
    sed 's/TAG$/*/g' |
    sed 's/TGA$/*/g' |
    sed 's/CGT/S/g' |
    sed 's/CGC/S/g' |
    sed 's/CGA/S/g' |
    sed 's/CGG/S/g' |
    sed 's/AGA/S/g' |
    sed 's/AGG/S/g' |
    sed 's/CGT/R/g' |
    sed 's/CGC/R/g' |
    sed 's/CGA/R/g' |
    sed 's/CGG/R/g' |
    sed 's/AGA/R/g' |
    sed 's/AGG/R/g'
}
convert=$(convert_cod)

echo "Converting codons to listed amino acids: "
echo $convert
```

Amino Acid conversation

Write a bash script that counts the number of ATG (starts), Serine (S), Arginine (R), and TAA, TAG, TGA (stops) from the example2.fasta file then converts them into amino acid M, S, R, and * for TAA, TAG, TGA for stop codon.

Remember that ATG encodes for methonine so they only count as start from the beginning of the sequence or the end for the stops. Serine and Arginine can be throughout the sequence.

OUTPU

```
ATGCTAAGGCTATCNNGACAACTGACTAAATAG
(base) docwhite@system76-pc:~/Desktop/BINF2111/data$ bash lab5_q4.sh example2.fasta
Number of starts: 7
Number of stops: 7
Number of Arginine: 7
Number of Serine: 5
Converting codons to listed amino acids:
>chr1_geneA MCTASCTATCTTGACAACTGACTGCC* >chr1_geneB MCTASCTATGTTGGCAACTGACTCCC* >chr1_geneC MCTASCTACCTTGACAACTGACTGGG* >chr1_geneD MAAASCTATCTTGACAACTGACTCCC* >chr1_geneX MCTASCTATCTTGATTTCTGACTTTT* >chr
1_geneY MGGGGGGCTATCTTGACAACTGACTGCG* >chr1_geneZ MCTASCTATCNNGACAACTGACTAAA*
```

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 (Python)

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

BASH - while loop

```
#!/bin/bash
x=1
while [ $x -le 5 ]
do
  echo "Welcome $x
times"
  x=$(( $x + 1 ))
done
```

BASH - while loop (one - liner)

```
x=1; while [ $x -le 5 ]; do echo "Welcome
$x times" $(( x++ )); done
```

BASH - functions

```
Function_name( ){
    command
}
```

Think of a function as a small script within a script.
It's a small chunk of code which you may call multiple times within your script.

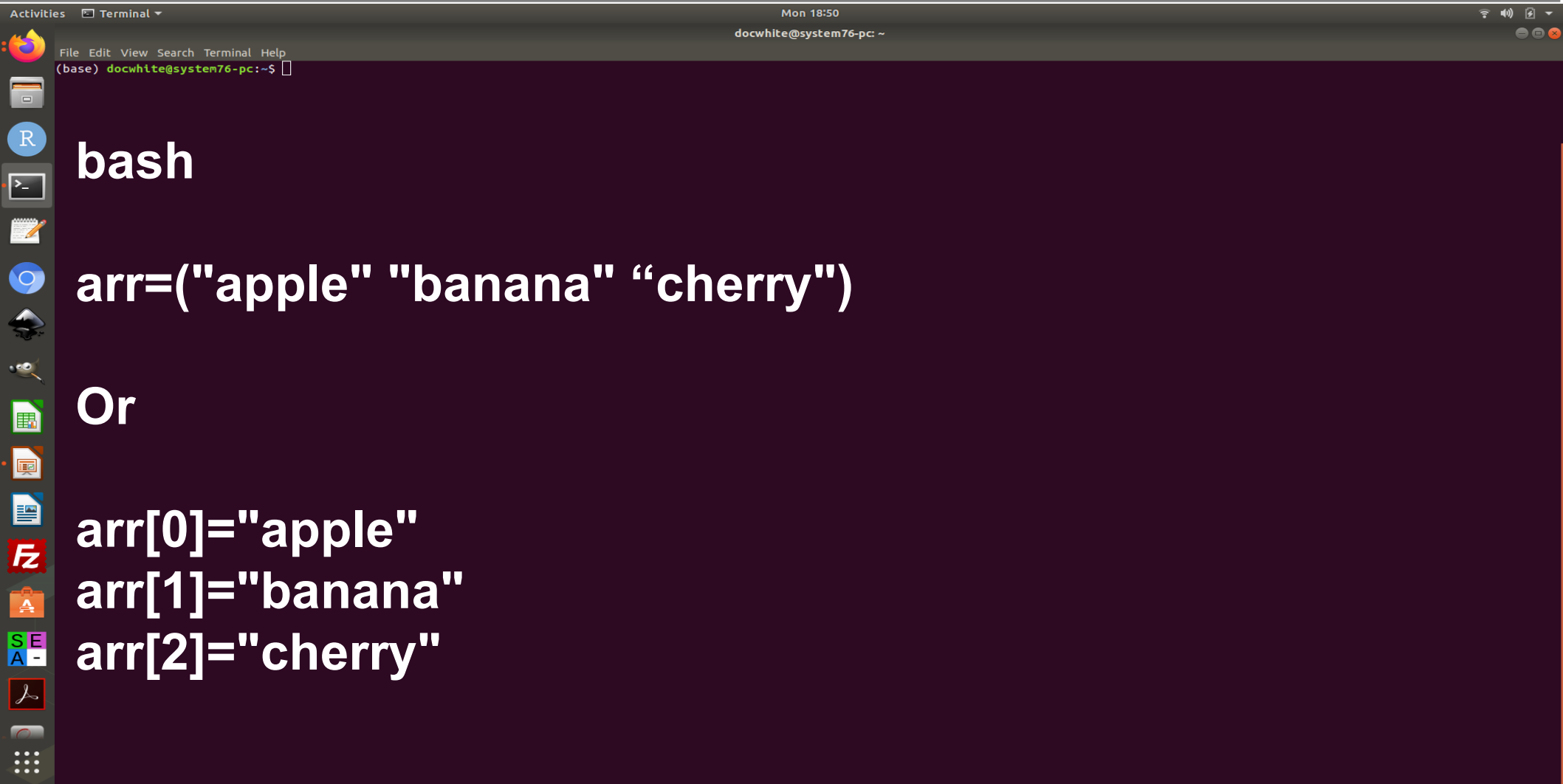
MY FAVORITE WAY! (There is another way)

BASH - functions

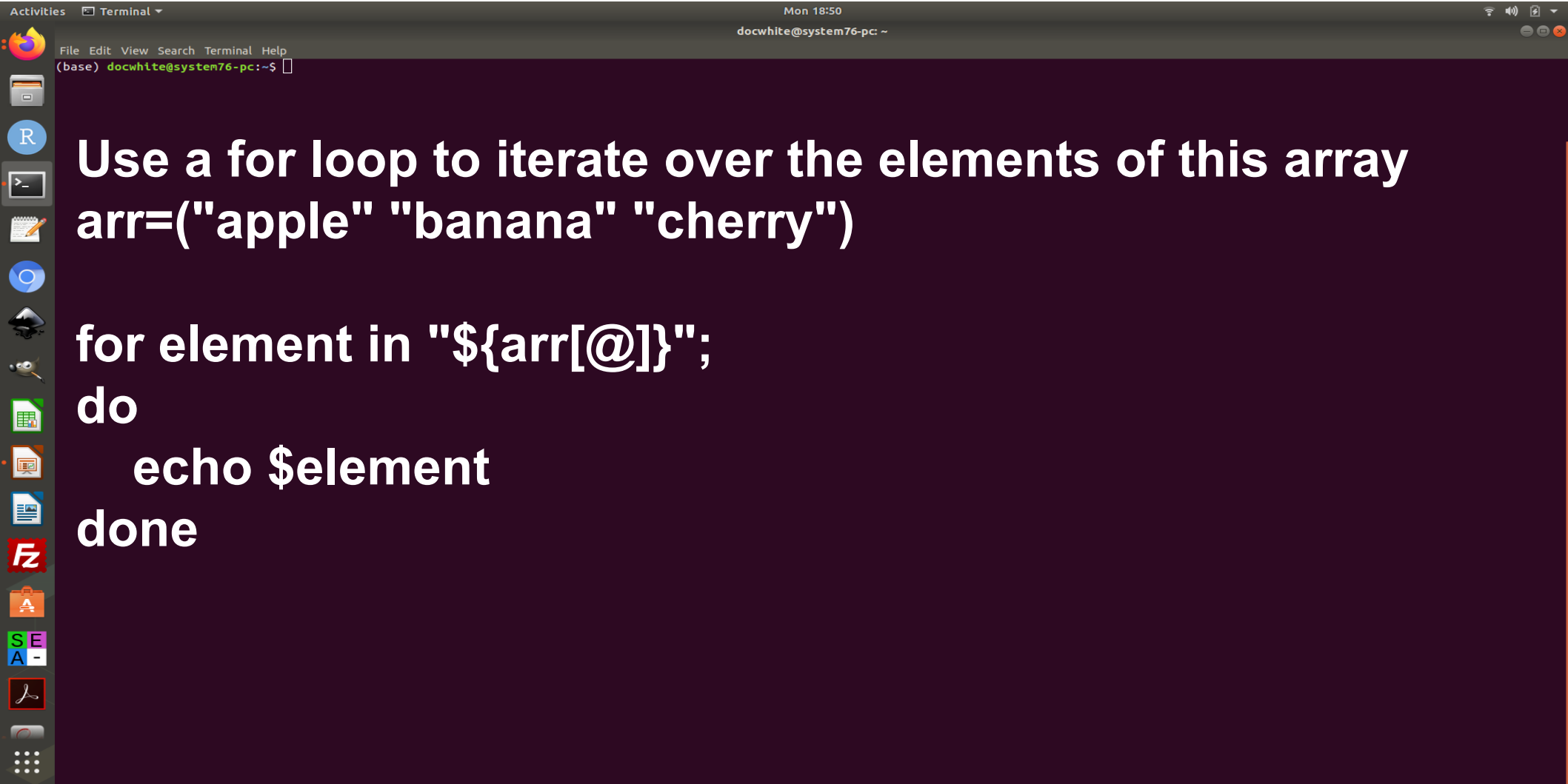
```
Function function_name( ){
    command
}
```

Not my favorite. But, you may like it?

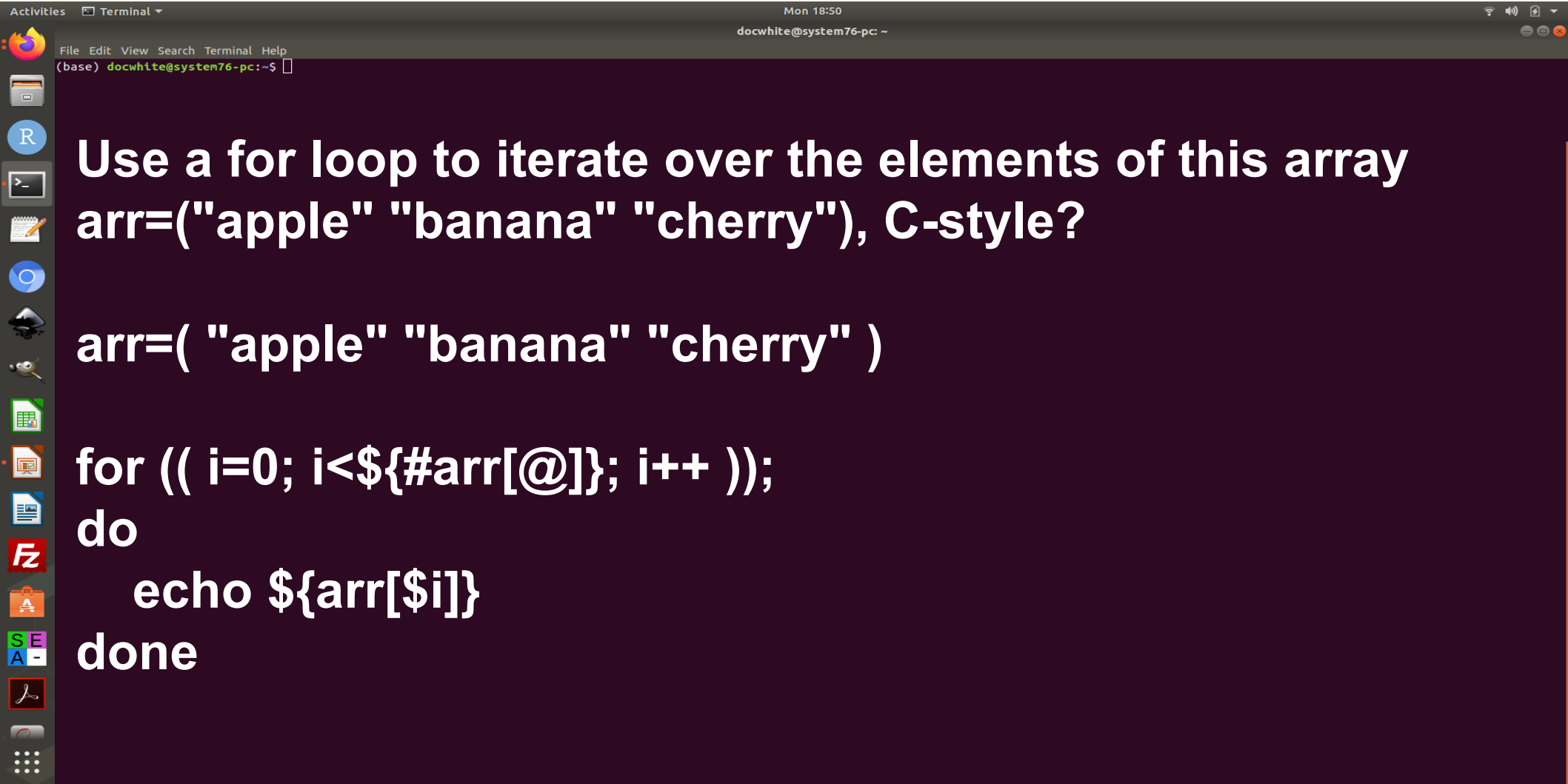
BASH arrays


Activities
Terminal
Mon 18:50
docwhite@system76-pc: ~
File Edit View Search Terminal Help
(base) docwhite@system76-pc:~$
bash
arr=("apple" "banana" “cherry")
Or
arr[0]="apple"
arr[1]="banana"
arr[2]="cherry"

BASH for loop in arrays



BASH for loop in arrays



Github – Basic Functions

git clone: Clones a repository into a newly created directory, creates remote-tracking branches for each branch in the cloned repository

git status: displays the state of the working directory and the staging area.

git pull: update the local version of a repository from a remote. By default, git pull does two things. Updates the current local working branch (currently checked out branch) Updates the remote tracking branches for all other branches

git commit: Create a new commit containing the current contents of the index and the given log message describing the changes.

git add: Updates the index using the current content found in the working tree, to prepare the content staged for the next commit.

git fetch: checks server for updates without ‘pulling’ them

Cloud services (compare)

Category	Service	amazon web services	Azure	Google Cloud Platform	IBM Cloud	ORACLE	Alibaba Cloud
Compute	**Shared Web hosting**		Azure shared App Services		Web hosting services		Web Hosting Simple Application Server
Compute	**Virtual Server**	Amazon EC2	Azure Virtual Machine	Compute Engine	Virtual Server Infrastructure (VSi)	Compute	Alibaba ECS
Compute	**Bare Metal Server**	Amazon EC2 Bare Metal Instance (Preview)	Azure Bare Metal Servers (Large Instance Only for SAP Hana)		Bare Metal Servers	Bare Metal Servers	ECS Bare Metal Instance
Compute	**Virtual Dedicated Host**	Amazon EC2 Dedicated Hosts		Sole Tenant Node (Beta)	Dedicated Virtual Servers Infrastructure (VSi)	Dedicated Compute Classic	Dedicated Host
Compute	**Container Registration Service**	Amazon EC2 Container Registry	Azure Container Registry	Container Registry	IBM Cloud Container Registry	Oracle Cloud Infrastructure Registry	Container Registry
Compute	**Container Management Service**	Amazon EC2 Container Service Amazon Elastic Container	Azure Kubernetes Service (AKS) Azure Container	Kubernetes Engine	IBM Cloud Kubernetes Service	Container Engine for Kubernetes (OKE)	Container Service Container Service for

What a slurm and bash shell is?

cluster_script.sh

```
#!/bin/bash

sleep 1
echo "Job# $1 from $(hostname)"
```

cluster-slurm_sbatch

```
#!/bin/bash

#SBATCH --partition=Centaurus          # Partition name (Centaurus or GPU)
#SBATCH --job-name=script_test         # Job name
#SBATCH --nodes=5                      # Number of total nodes (computers on the cluster)
#SBATCH --mem=1gb                      # Memory per node
#SBATCH --time=0:10:00                 # Time limit hrs:min:sec OR days-hrs
#SBATCH --output=out-%x-%j.log         # Standard output and error log


SECONDS=0

for i in $(seq 5)
do
    echo "Starting job: $i"
    srun --nodes=1 --exclusive ./script.sh $i &
done
wait

echo "Runtime is: $SECONDS seconds"
```

Quiz 14

- On canvas now