"IBS 574 - Computational Biology & Bioinformatics"
Spring 2018, Tuesday (02/01), 2.00-4.00PM

Linux shell & shell scripting - II

Ashok R. Dinasarapu Ph.D
Scientist-Bioinformatics, Dept. of Human Genetics
Emory University, Atlanta

What is Shell scripting?

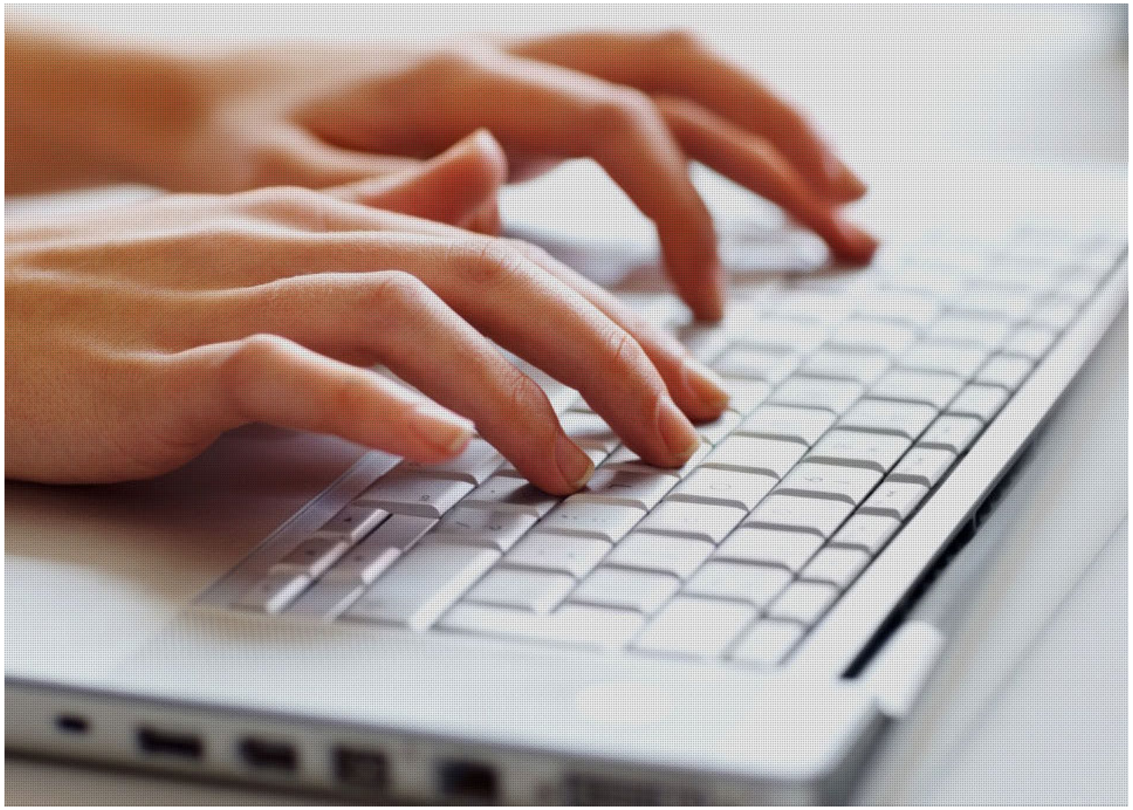
- A script is a collection of commands stored in a file.
- Shell is a command-line interpreter.
- To execute a shell script, ex. “hello.sh”

  Usage:

```
./hello.sh
. ./hello.sh
/home/user_name/script/hello.sh
bash hello.sh (or sh hello.sh)
```

Easiest way to do this is …

Start typing!



Connecting via SSH to your server

ssh *user_name*@blnx1.emory.edu

Alternatively, https://blnx1.emory.edu:22443/

```
user_name@blnx1:~$
```

~ means your home dir (/home/*user_name*)

SSH allows you to connect to your server securely and perform Linux command-line operations.

“mkdir” & directory structure

- Create directories from your home directory (i.e /home/user_name)

Usage:
```
mkdir -p project/{data,script,out,log}
```

project/data

project/script

project/out

project/log

Usage:
```
cd project/script
```

Create/Edit text files

Choose a text editor: emacs, Vim

Usage: `vi hello.sh`

INSERT mode:
press keys like `i` OR `a` & start typing.

"i" will let you insert text just before the cursor.
"I" inserts text at the beginning of the current line.
"a" will let you insert text just after the cursor, and
"A" will let you type at the end of the current line.

Create/Edit text files

Type the following text:

```
#!/bin/sh
# My first script
echo "Hello World!"
```

“**#!/bin/sh**” a special clue given to the shell indicating what program is used to interpret the script.

Create/Edit text files

Type the following text:

```
#!/bin/bash
#!/usr/bin/perl
#!/usr/bin/Rscript
#!/usr/bin/env python
```

```
#!/bin/sh
# My first script
echo "Hello World!"
```

View symbolic link using **ls –la** /bin/sh

“**#!/bin/sh**” a special clue given to the shell indicating what program is used to interpret the script.

Create/Edit text files

SAVE mode:

press `esc` key AND

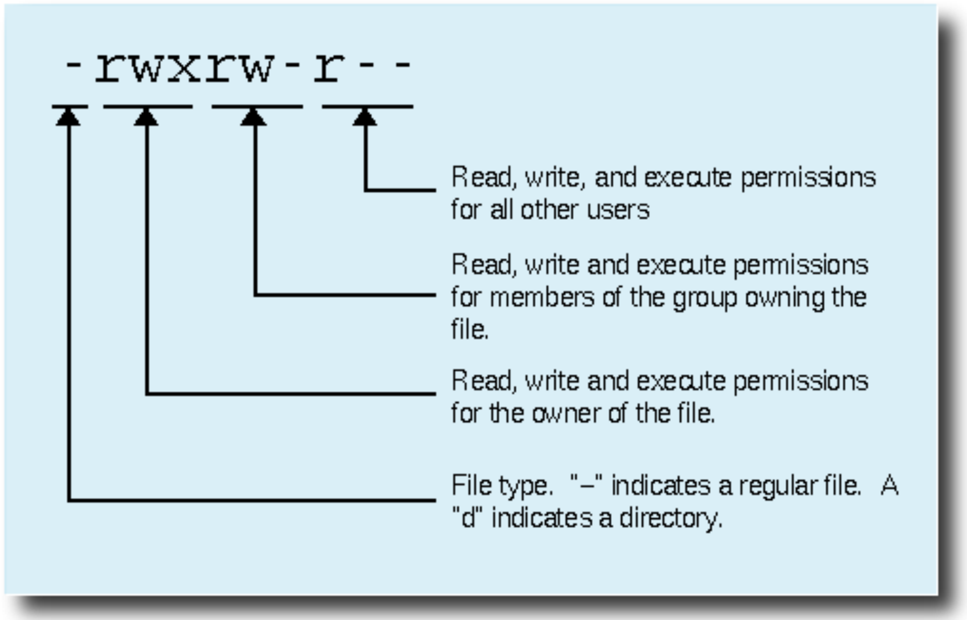
`:q!` for not to save OR

`:x` to save all typed content.

"chmod"

- Change the permissions of files
  - **R**ead (r), **w**rite (w), and e**x**ecute (x)
  - 3 types of users (user, group & other)

Usage: ls -l


-rwxrw-r--
Read, write, and execute permissions for all other users
Read, write and execute permissions for members of the group owning the file.
Read, write and execute permissions for the owner of the file.
File type. "–" indicates a regular file. A "d" indicates a directory.

"make file executable"

- **To make it executable**

  Usage: chmod +**rwx**r-x**r-x** hello.sh

  Usage: `chmod +x hello.sh`

  Usage: `./hello.sh`

- **To make it un-executable**

  Usage: `chmod -x hello.sh`

“make file executable”

- **To make it executable** +755

  Usage: chmod +**rwx**r-x**r-x** hello.sh

  Usage: `chmod +x hello.sh`

  Usage: `./hello.sh`

- **To make it un-executable**

  Usage: `chmod -x hello.sh`

Variables in Linux/Shell

Usage: `vi hello.sh`

```
#!/bin/bash
# My first script
i=22
echo "Hello, I am " $i
# echo "Hello, I am  $i"
```

Usage: `./hello.sh`

Variables in Linux/Shell

Usage: `vi hello.sh`

```
#!/bin/bash
# My first script
i=22
j="Hello, I am ”
echo $j $i
```

Usage: `./hello.sh`

Variables in Linux/Shell

Usage: `vi hello.sh`

```
#!/bin/bash
# My first script
i=22
j=${i}0
echo $j
```

the {} in ${} are useful if you want to expand the variable

Usage: `./hello.sh`

Conditions in Linux/Shell

Usage: `vi hello.sh`

```
#!/bin/bash
# My first script
PASS="test1234"
if [ $PASS == "test1234" ]; then
    echo "Correct pass word!!"
fi
```

Usage: `./hello.sh`

Remember that the spacing is very important in the if statement.

Conditions in Linux/Shell

Usage: `vi hello.sh`

```
#!/bin/bash
# My first script

PASS="test123"

if [ $PASS=="test1234" ]; then
     echo "Correct pass word!!"
else
     echo "enter correct pass word!!"
fi
```

Usage: `./hello.sh`

“for” loop in Linux/Shell

Usage: `vi hello.sh`

```
#!/bin/bash
# My first script
for i in {1..10}; do
    echo $i
    # echo ${i}a
done
```

Usage: `./hello.sh`

"**Array**" in Linux/Shell

Usage: `vi hello.sh`

```
#!/bin/bash
arr=('A' 'B' 'C' 'D' 'E')
for i in {0..4}; do
    echo $i
    echo ${arr[$i]}
done
```

Usage: `./hello.sh`

“for” loop in Linux/Shell

Usage: `vi hello.sh`

```
#!/bin/sh
# My first script
for i in 1 2 3 x y z
do
    echo $i
done
```

loop indices doesn't have to be just numbers!!!

{1..3} x y z

Use **bash**

Usage: `./hello.sh`

“while” loop in Linux/Shell

Usage: vi hello.sh

```
#!/bin/bash
# My first script

i=0
while [ $i -le 5 ]; do
    # echo “before $i”
    i=$(( $i+1 ))
    echo “after $i”
done
```

Usage: ./hello.sh

Functions in Linux/Shell

Usage:
```
vi hello.sh
```

```
#!/bin/bash
# function definition
add_user()
{
      USER=$1
      PASS=$2
      echo “Passwd $PASS created for $USER on $(date)”
}
# function call
echo $(add_user bob letmein)
```

Usage:
```
./hello.sh
```

Functions in Linux/Shell

Usage: `vi hello.sh`

```
#!/bin/bash
# function definition
add_user()
{
      # USER=$1
      # PASS=$2
      echo “Passwd $1 created for $2 on $3”
}
# function call
echo $(add_user bob letmein “$(date)”)
```

Usage: `./hello.sh`

download a fastq file

Usage: `cd project/data`

wget
https://github.com/CGATOxford/UMI-tools/releases/download/v0.2.3/example.fastq.gz

- **View**

  Usage: `zcat example.fastq.gz|head`

- **Check file size**

  Usage: `ls -lh example.fastq.gz`

- **Count number of sequences in a fastq file**

  Usage: `grep -c "^>" example.fastq.gz`

“Calculate the length of reads”

Create the following executable file at **project/script**

Usage:
```
vi fastq.sh
```

```
#!/bin/bash
# cd project/log  (to run this script from log directory)
zcat ../data/example.fastq.gz | \
awk ‘{if(NR%4==2) print length($1)}’ > ../out/length.txt
```

Usage:
```
../script/fastq.sh
```

"Plot - Histogram"

Create the following executable file at **project/script**

Usage:
```
vi hist.R
```

```
#!/usr/bin/Rscript
# cd project/log  (to run this script from log dir)
t.dat <- read.table('../out/length.txt')
jpeg('../out/rplot.jpg')
hist(t.dat[,1])
dev.off()
```

Usage: ../script/hist.R

“Calculate the length of reads”

Create the following executable file at **project/script**

Usage:
```
vi fastq.sh
```

```
#!/bin/bash
# cd project/log  (to run this script from log directory)
zcat ../data/example.fastq.gz | \
awk ‘{if(NR%4==2) print length($1)}’ > ../out/length.txt
# include R script and run
../script/hist.R
```

Usage:
```
../script/fastq.sh
```

“Calculate the length of reads”

Create the following executable file at **project/script**

Usage: `vi fastq.sh`

```
#!/bin/bash
# cd project/log  (to run this script from log directory)

zcat ../data/example.fastq.gz | \
awk '{if(NR%4==2) print length($1)}' > ../out/length.txt

# include R script and run

../script/hist.R
```

Usage: `../script/fastq.sh 2>error.log`

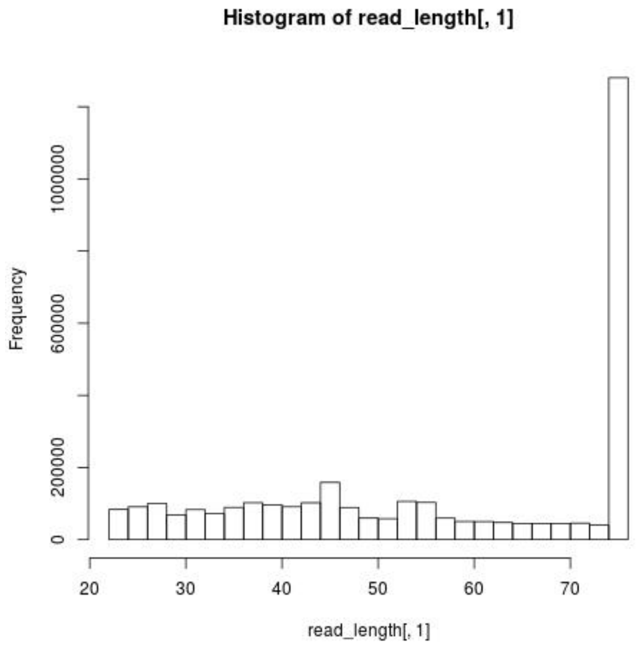
Redirect error output to a file

Open/View Image

- Login as **interactive mode**

Usage:
```
ssh -Y user_name@blnx1.emory.edu
```

Usage:
```
xdg-open rplot.jpg
```



-X Enables X11 forwarding
-Y Enables trusted X11 forwarding

X11 is the X Window System

AWK

- AWK is an interpreted programming language & designed for text processing.

  The word "AWK" is derived from the initials of the language's three developers: A. Aho, B. W. Kernighan and P. Weinberger.

- Following are the variants of AWK
  - AWK - the (very old) original from AT&T
  - NAWK - A newer, improved version from AT&T
  - GAWK – It is GNU AWK. The Free Software foundation's version

Basic Structure of AWK

pattern { *action* }

- The pattern specifies when the action is performed.
- By default, AWK execute commands on every line.
- AWK reads a line from the input stream (file, pipe, or STDIN)
- This process repeats until the file reaches its end.

Basic Structure of AWK

pattern { *action* }

- Two other important patterns are specified by the keywords "BEGIN" and "END”

```
BEGIN   { print "START" }
pattern { print }
  END   { print "STOP"  }
```

- **BEGIN block** executes only once. This is good place to initialize variables.
- The **body block** applies AWK commands on every input line.
- The **END block** executes at the end of the program.

File processing/Printing records

Choose a text editor: emacs, Vim

Usage:

```
vi marks.txt
```

```
1   Amit    Physics    80
2   Ralf    Maths      90
3   Shyam   Biology    87
4   Kedar   English    85
5   Ashok   History    89
```

```
awk '{print}' < marks.txt
```

```
awk 'BEGIN{printf "SNo\tName\tSub\tMarks\n"}{print}' < marks.txt
```

Printing Fields (Columns)

Choose a text editor: emacs, Vim

Usage:

```
vi marks.txt
```

```
1    Amit    Physics    80
2    Ralf    Maths      90
3    Shyam   Biology    87
4    Kedar   English    85
5    Ashok   History    89
```

```
awk '{print}' < marks.txt
awk '{print $0}' < marks.txt
awk '{print $1,$2,$3,$4}' < marks.txt              (space)
awk '{print $1"\t"$2"\t"$3"\t"$4}' < marks.txt     (tab)
```

Printing Columns by Pattern

Choose a text editor: emacs, Vim

Usage:

```
vi marks.txt
```

```
1   Amit    Physics   80
2   Ralf    Maths     90
3   Shyam   Biology   87
4   Kedar   English   85
5   Ashok   History   89
```

```
awk ’/E/ {print}' < marks.txt
awk ’/E/' < marks.txt
awk '/A/ {print $2"\t"$4}' < marks.txt      (tab)
```

Count by Pattern match

Choose a text editor: emacs, Vim

Usage:

```
vi marks.txt
```

1	Amit	Physics	80
2	Ralf	Maths	90
3	Shyam	Biology	87
4	Kedar	English	85
5	Ashok	History	89

```
awk
'BEGIN{i=0}
 /A/ {i+=1}
 END{print i}
' < marks.txt
```

```
awk 'BEGIN{i=0} /A/ {i+=1} END{print i}' < marks.txt
```

Count by Pattern match

Choose a text editor: emacs, Vim

Usage:

```
vi marks.txt
```

1	Amit	Physics	80
2	Ralf	Maths	90
3	Shyam	Biology	87
4	Kedar	English	85
5	Ashok	History	89

```
awk \
'BEGIN{i=0} \
 /A/ {i+=1} \
 END{print i} \
' < marks.txt
```

awk is one-liner

```
awk 'BEGIN{i=0} /A/ {i+=1} END{print i}' < marks.txt
```

Count by Pattern match

Choose a text editor: emacs, Vim

Usage:

```
vi marks.txt
```

```
#!/bin/bash
awk \
'BEGIN{i=0} \
 /A/ {i+=1} \
 END{print i} \
' < marks.txt
```

1	Amit	Physics	80
2	Ralf	Maths	90
3	Shyam	Biology	87
4	Kedar	English	85
5	Ashok	History	89

```
./marks.sh
```

Count by Pattern match

Choose a text editor: emacs, Vim

Usage:

```
vi marks.txt
```

1	Amit	Physics	80
2	Ralf	Maths	90
3	Shyam	Biology	87
4	Kedar	English	85
5	Ashok	History	89

```
#!/usr/bin/awk –f
BEGIN{i=0} \
 /A/ {i+=1} \
 END{print i}
```

```
awk -f marks.awk < marks.txt
```

Standard AWK variables

env

```
XDG_SESSION_ID=38
TERM=xterm-256color
SHELL=/bin/bash
SSH_CLIENT=10.110.52.30 28952 22
SSH_TTY=/dev/pts/8
USER=adinasarapu
…
```

```
awk 'BEGIN {print ENVIRON["USER"]} '
awk 'BEGIN {print ENVIRON["SHELL"]} '
awk 'BEGIN {print ENVIRON["HOSTNAME"]}’
```

```
awk 'END { print FILENAME }' marks.txt
```

(Please note that FILENAME is undefined in the BEGIN block)

Standard AWK variables

FS, **OFS**, RS, ORS, NR and NF

FS: Input Field Separator (default is space)
OFS: Output Field Separator (default is space)

```
env | awk 'BEGIN{FS="=";}{print $1}’

env | awk 'BEGIN{FS="="; OFS=":"}{print $1,$2}'
```

Standard AWK variables

FS, **OFS**, RS, ORS, NR and NF

FS: Input Field Separator (default is space)
OFS: Output Field Separator (default is space)

```
awk '{print $1,$2,$3,$4}' < marks.txt                  (space)
awk '{print $1"\t"$2"\t"$3"\t"$4}' < marks.txt    (tab)

awk 'BEGIN{FS="\t"; OFS="\t"}{print $1,$2,$3,$4}' < marks.txt
```

Standard AWK variables

FS, **OFS**, RS, ORS, NR and NF

FS: Input Field Separator (default is space)
OFS: Output Field Separator (default is space)

```
awk '{print $1,$2,$3,$4}' < marks.txt                 (space)
awk '{print $1"\t"$2"\t"$3"\t"$4}' < marks.txt    (tab)

awk 'BEGIN{FS="\t"; OFS="\t"}{print $1,$2,$3,$4}' < marks.txt
```

```
awk -F'\t' 'BEGIN{OFS="\t"}{print $1,$2,$3,$4}' < marks.txt
```

Standard AWK variables

FS, OFS, **RS**, **ORS**, NR and NF

RS: Input Record Separator (default is Newline)
ORS: Output Record Separator (default is Newline)

```
awk '{print $1,$2,$3,$4}' < marks.txt                (space)
awk '{print $1"\t"$2"\t"$3"\t"$4}' < marks.txt    (tab)

awk 'BEGIN{FS="\t"; OFS="\t"}{print $2,$4}' < marks.txt
```

```
awk 'BEGIN{OFS=":"; ORS=";"}{print $2,$4} END{print "\n"}' < marks.txt
```

Standard AWK variables

FS, OFS, RS, ORS, **NR** and **NF**

NR: Record number

NF: Number of Fields in a Record

```
cat marks.txt | wc –l        (Number of lines/rows)
cat marks.txt | wc –w        (Number of words)
```

```
cat marks.txt | awk '{print NR,"->",NF} '

cat marks.txt | awk 'NR<4'                      (get first 3 records)

cat marks.txt | awk '{print NR}' | tail -1  (number of records)

cat marks.txt | awk '{ total = total + NF } END { print total }'
```

AWK - Control Flow

- AWK provides conditional statements to control the flow of a program

if or if else

```
#!/bin/bash
awk 'BEGIN {
  num =10;
  if (num % 2 == 0)
     printf "%d is even number.\n", num;
   else
     printf "%d is odd number.\n", num
}'
```

AWK - Control Flow

- AWK provides conditional statements to control the flow of a program

for loop

in is used while accessing array elements

```
#!/bin/bash
awk 'BEGIN {
   arr[0] = 11;
   arr[1] = 22;
   arr[2] = 33;
   for(i in arr) printf "%d is a number.\n", arr[i]
}'
```

```
awk 'BEGIN { arr[0]=11; arr[1]=22; for(i in arr) printf "%d is a number.\n", arr[i] }'
```

AWK - Control Flow

- A FASTQ file normally uses four lines per sequence

```
zcat example.fastq.gz | head

@SRR2057595.7
CAGGTTCAATCTCGGTGGGACCTC
+SRR2057595.7
1=DFFFFHHHHHJJJFGIJIJJIJ
@SRR2057595.9
TTGGTTCAATCTGATGCCCTCTTCTGGTGCATCTGAAGACAGCTACAGTGTACTTAGATATAATAAATAAATCTT
+SRR2057595.9
4=DFDBDHHFHHIGGEHJGGIHGHGGCAFCHGIGEHIJJJJIJJJIHIIIIIIJIIIIIGHIIGGIJGIIJIIJ@
@SRR2057595.14
TGGGTTAATGCGGCCCCGGGTTCCTCCCGGGGCTACGCCTGTCTGAGCGTCGCT
...
```

Sequence ID

Sequence

Line 1 begins with a '@' character and is followed by a sequence identifier
Line 2 is the raw sequence letters
Line 3 begins with a '+' character and is *optionally* followed by the same sequence identifier
Line 4 encodes the quality values for the sequence in Line 2

AWK - Control Flow

- A FASTQ file normally uses four lines per sequence

```
zcat example.fastq.gz | head

@SRR2057595.7                          Sequence ID
CAGGTTCAATCTCGGTGGGACCTC               Sequence
+SRR2057595.7
1=DFFFFHHHHHJJJFGIJIJJIJ
@SRR2057595.9
TTGGTTCAATCTGATGCCCTCTTCTGGTGCATCTGAAGACAGCTACAGTGTACTTAGATATAATAAATAAATCTT
+SRR2057595.9
4=DFDBDHHFHHIGGEHJGGIHGHGGCAFCHGIGEHIJJJJIJJJIHIIIIIJIIIIIGHIIGGIJGIIJIIJ@
@SRR2057595.14
TGGGTTAATGCGGCCCCGGGTTCCTCCCGGGGCTACGCCTGTCTGAGCGTCGCT
…
```

```
zcat example.fastq.gz | awk '{if(NR%4==2) print}' | head -20
zcat example.fastq.gz | awk '{if(NR%4==2) print $1}' | head -20
zcat example.fastq.gz | awk '{if(NR%4==2) {print substr($0,1,10)}}' | head -20
zcat example.fastq.gz | awk 'END{print NR/4}’
```

AWK - Control Flow

# Separate (filter) FASTQ reads based on their length

```
#!/bin/bash
zcat example.fastq.gz | \
	awk 'NR%4 == 1 {ID=$0}
		NR%4==2 {SEQ=$0}
		NR%4==3 {PLUS=$0}
		NR%4==0 {QUAL=$0}
		{
			seqlen=length(SEQ)
			qualen=length(QUAL)
			if(seqlen==qualen&&seqlen>=75)
			print ID"\n"SEQ"\n"PLUS"\n"QUAL | \
			"gzip > filtered.fastq.gz"
		}
	'
```

AWK - Control Flow

# Separate (filter) FASTQ reads based on their length

```
#!/bin/bash
zcat example.fastq.gz | \
    awk 'NR%4 == 1 {ID=$0}
        NR%4==2 {SEQ=$0}
        NR%4==3 {PLUS=$0}
        NR%4==0 {QUAL=$0}
        {
            seqlen=length(SEQ)
            qualen=length(QUAL)
            if(seqlen==qualen&&seqlen>=75)
            print ID"\n"SEQ"\n"PLUS"\n"QUAL | \
            "gzip > filtered.fastq.gz"
        }
    '
```

```
echo $((1%4))  1
echo $((2%4))  2
echo $((3%4))  3
echo $((4%4))  0
echo $((5%4))  1
echo $((6%4))  2
echo $((7%4))  3
echo $((8%4))  0
…
…
```

Practice Makes Perfect

Thank you

String Comparison Operators

Operator	Description	Example
= or ==	Is Equal To	if ["$1" == "$2"]
!=	Is Not Equal To	if ["$1" != "$2"]
>	Is Greater Than (ASCII comparison)	if ["$1" > "$2"]
>=	Is Greater Than Or Equal To	if ["$1" >= "$2"]
<	Is Less Than	if ["$1" < "$2"]
<=	Is Less Than Or Equal To	if ["$1" <= "$2"]
-n	Is Not Null	if [-n "$1"]
-z	Is Null (Zero Length String)	if [-z "$1"]

Integer Comparison Operators

Operator	Description	Example
-eq	Is Equal To	if [$1 -eq 200]
-ne	Is Not Equal To	if [$1 -ne 1]
-gt	Is Greater Than	if [$1 -gt 15]
-ge	Is Greater Than Or Equal To	if [$1 -ge 10]
-lt	Is Less Than	if [$1 -lt 5]
-le	Is Less Than Or Equal To	if [$1 -le 0]
==	Is Equal To	if (($1 == $2))
!=	Is Not Equal To	if (($1 != $2))
<	Is Less Than	if (($1 < $2))
<=	Is Less Than Or Equal To	if (($1 <= $2))
>	Is Greater Than	if (($1 > $2))
>=	Is Greater Than Or Equal To	if (($1 >= $2))