


Bash scripts - pt. 2A

▼ Type	 Lecture
📅 Date	@February 9, 2022
☰ Lecture #	1
🔗 Lecture URL	https://youtu.be/8YmNovNo6NE
🔗 Notion URL	https://21f1003586.notion.site/Bash-scripts-pt-2A-f7361a999fa344499488a73d6db381b1
# Week #	6

Debugging

```
set -x
./myscript.sh
```

Prints the command before executing it

Place the `set -x` inside the script

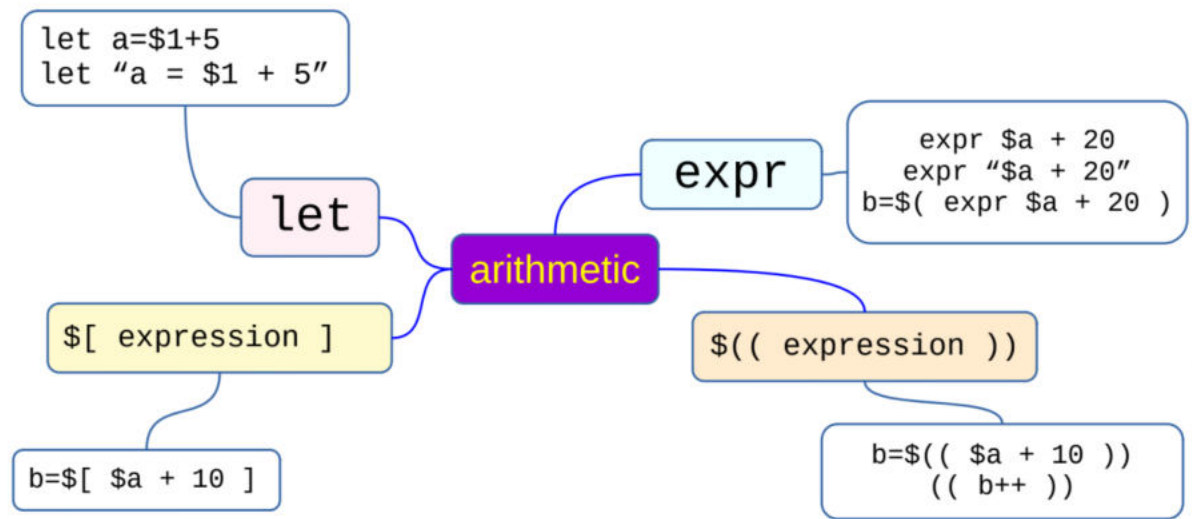
```
bash -x ./myscript.sh
```

Combining conditions

```
[ $a -gt 3 ] && [ $a -lt 7 ]
```

```
[ $a -le 3 ] || [ $a -ge 7 ]
```

Shell arithmetic



expr command operators

<code>a + b</code>	Return arithmetic sum of a and b
<code>a - b</code>	Return arithmetic difference of a and b
<code>a * b</code>	Return arithmetic product of a and b
<code>a / b</code>	Return arithmetic quotient of a divided by b
<code>a % b</code>	Return arithmetic remainder of a divided by b
<code>a > b</code>	Return 1 if a greater than b; else return 0
<code>a >= b</code>	Return 1 if a greater than or equal to b; else return 0
<code>a < b</code>	Return 1 if a less than b; else return 0
<code>a <= b</code>	Return 1 if a less than or equal to b; else return 0
<code>a = b</code>	Return 1 if a equals b; else return 0

a b	Return a if neither argument is null or 0; else return b
a & b	Return a if neither argument is null or 0; else return 0
a != b	Return 1 if a is not equal to b; else return 0
str : reg	Return the position upto anchored pattern match with BRE str
match str reg	Return the pattern match if reg matches pattern in str
substr str n m	Return the substring m chars in length starting at position n
index str chars	Return position in str where any one of chars is found; else return 0
length str	Return numeric length of string str
+ token	Interpret token as string even if its a keyword
(exprn)	Return the value of expression exprn

```
#!/bin/bash

if [ $# -lt 2 ]; then
    echo "Use 2 natural numbers as arguments";
    exit 1;
fi;

regex='^[0-9]+$'
if ! [[ $1 =~ $regex ]]; then
    echo "$1 is not a natural number";
    exit 1;
fi;

if ! [[ $2 =~ $regex ]]; then
    echo "$2 is not a natural number";
    exit 1;
fi;

let a=$1*$2;
echo "Product a is $a";
(( a++ ));
echo "Product a incremented is $a";

let b=$1**$2;
echo "Power is $b";

c=$(( $1 + $2 + 10 ));
echo "sum + 10 is $c";

d=$((expr $1 + $2 + 20));
echo "sum + 20 is $d";

f=$(( $1 * $2 * 2 ));
echo "Product times 2 is $f";
```

```

#!/bin/bash
# The following line is for debugging
# set -x;

# Code starts here
a=256;
b=4;
c=3;

ans=$( expr $a + $b );
echo $ans;

ans=$( expr $a - $b );
echo $ans;

ans=$( expr $a \* $b );
echo $ans;

ans=$( expr $a / $b );
echo $ans;

ans=$( expr $a % $c );
echo $ans;

ans=$( expr $a \> $b );
echo $ans;

ans=$( expr $a \>= $b );
echo $ans;

ans=$( expr $a \< $b );
echo $ans;

ans=$( expr $a \<= $b );
echo $ans;

ans=$( expr $a = $b );
echo $ans;

ans=$( expr $a != $b );
echo $ans;

ans=$( expr $a \| $b );
echo $ans;

ans=$( expr $a \& $b );
echo $ans;

str="octavio version as in Jan 2022 is 6.4.0";
reg="[oO]ctav[aeiou]*";
ans=$( expr "$str" : $reg );
echo $ans;

ans=$( expr substr "$str" 1 6 );
echo $ans;

```

```
ans=$( expr index "$str" "vw" );
echo $ans;

ans=$( expr length "$str" );
echo $ans;
```

heredoc feature

When writing shell scripts you may be in a situation where you need to pass a multiline block of text or code to an interactive command, such as `tee`, `cat`, or `sftp`

In `bash` and other shells like `zsh`, a Here document (`heredoc`) is a type of redirection that allows you to pass multiple lines of input to a command.

This is what a general syntax looks like

```
[COMMAND] <<[-] 'DELIMITER'
    HERE-DOCUMENT
DELIMITER
```

```
a=2.5
b=3.2
c=4
d=$(bc -l << EOF
scale = 5
($a+$b)^$c
EOF
)
echo $d
```

1055.6001

Marker designation need not be EOF

```
a=2.5
b=3.2
c=4
d=$(bc -l <<- ABC
    scale = 5
    ($a+$b)^$c
    ABC
)
echo $d
```

A hyphen tells bash to ignore leading tabs

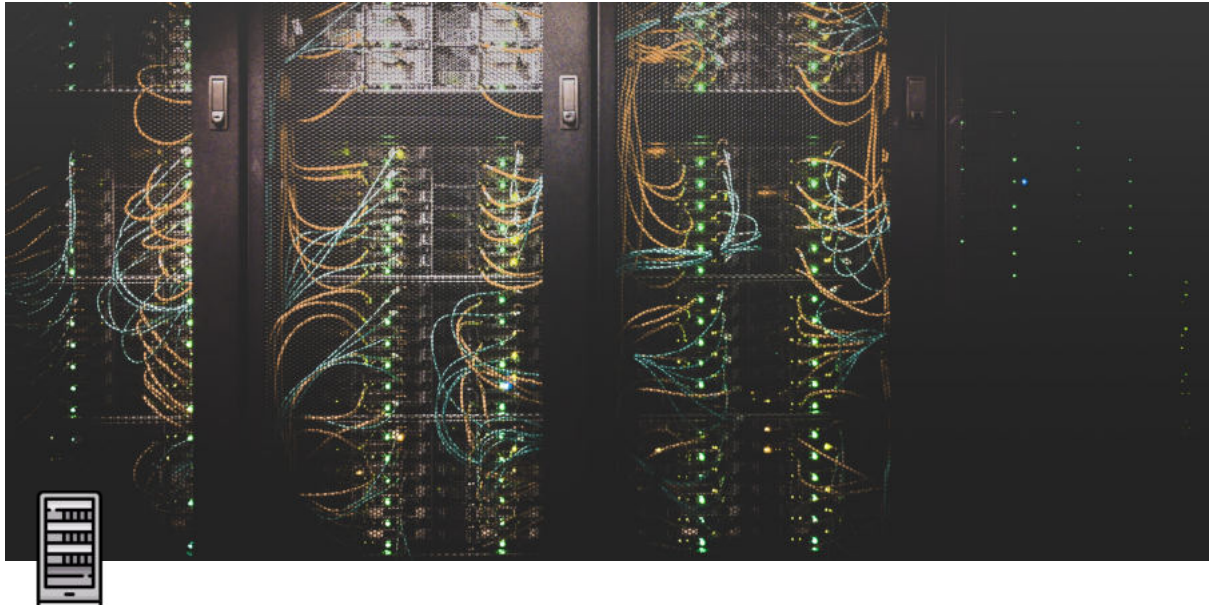
Notice no-indent (left) vs indentation (right) in the above example

```
#!/bin/bash


# set -x;
echo "path is set as $PATH";
i=0;
IFS=;;

for n in $PATH; do
    echo "$i $n";
    (( i++ ));
done;
```





Bash scripts - pt. 2B

▼ Type	 Lecture
📅 Date	@February 9, 2022
☰ Lecture #	2
🔗 Lecture URL	https://youtu.be/n56JIC15DBo
🔗 Notion URL	https://21f1003586.notion.site/Bash-scripts-pt-2B-caebca1e09d24acfba760344452db02d
# Week #	6

if-elif-else-fi loop

```

if condition1
then
    commandset1
else
    commandset2
fi

```

```

if condition1
then
    commandset1
elif condition2
then
    commandset2
elif condition3
then
    commandset3
else
    commandset4
fi

```

```

#!/bin/bash

if [ $# -gt 2 ]; then
    echo "More than 2 arguments";
elif [ $# -gt 1 ]; then
    echo "More than 1 argument";
elif [ $# -gt 0 ]; then
    echo "Not enough arguments";
else
    echo "Arguments required";
fi;

```

case statement options

```

case $var in
    op1)
        commandset1;;
    op2 | op3)
        commandset2;;
    op4 | op5 | op6)
        commandset3;;
    *)
        commandset4;;
esac

```

commandset4 is the default for values of **\$var** not matching what are listed

```

#!/bin/bash

echo "What is your favourite image processor?";
read pname;

```

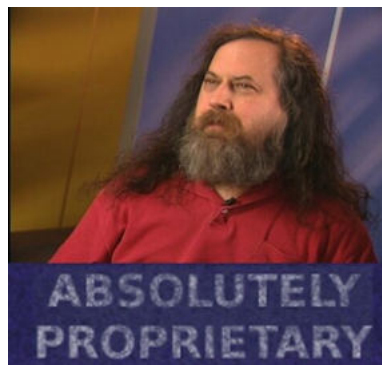


```

case $pname in
    [gG]imp | inkscape)
        echo "Good choice";
        ;;
    [aA]dobe*)
        echo "Absolutely proprietary and costs a lot";
        ;;
    imagej)
        echo "Measuring things on the image?";
        ;;
    *)
        echo "$pname is a new find for me";
        ;;
esac;

```

When you enter "adobe"



C-style `for` loop → one variable

```

begin=1
finish=10
for (( a = $begin; a < $finish; a++ ))
do
    echo $a
done

```

```

#!/bin/bash

begin=1;
finish=10;

```

```
for (( a = $begin; a < $finish; a++ )); do
    b=$(( a**2 ));
    echo $b;
done;
```

C-style **for** loop → two variables

```
begin1=1
begin2=10
finish=10
for (( a=$begin1, b=$begin2; a < $finish; a++, b-- ))
do
    echo $a $b
done
```

NOTE: only one condition to close out the for loop

```
#!/bin/bash

begin1=1;
begin2=20;
finish=10;

for (( a = $begin1, b = $begin2; a < $finish; a++, b-- )); do
    c=$(( a**2 ));
    d=$(( b**2 ));
    echo $c $d;
done;
```

Processing output of a loop

```
filename=tmp.$$
begin=1
finish=10
for (( a = $begin; a < $finish; a++ ))
do
    echo $a
done > $filename
```

NOTE: Output of the loop is re-directed to the tmp file

```
#!/bin/bash

filename=largefile.txt;
if [ -e $filename ]; then
    echo "file $filename exists";
    exit 1;
fi;

i=1;
while [ $i -lt 10 ]; do
    echo "$i ${i+1}";
    (( i++ ));
done > $filename;

echo "file $filename written";
ls -l $filename;
```

break

To break out of a loop

```
n=10
i=0
while [ $i -lt $n ]
do
    echo $i
    (( i++ ))
    if [ $i -eq 5 ]
    then
        break
    fi
done
```

break out of inner loop

```

n=10
i=0
while [ $i -lt $n ]
do
    echo $i
    j=0
    while [ $j -le $i ]
    do
        printf "$j "
        (( j++ ))
        if [ $j -eq 7 ]
        then
            break 2
        fi
    done
    (( i++ ))
done

```

break out of outer loop

```

0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
0 1 2 3 4 5 6
0 1 2 3 4 5 6

```

`break 2` refers to the outer loop as seen from the nesting

`continue`

Opposite of `break`

```

n=9
i=0
while [ $i -lt $n ]
do
    printf "\n loop $i:"
    j=0
    (( i++ ))
    while [ $j -le $i ]
    do
        (( j++ ))
        if [ $j -gt 3 ] && [ $j -lt 6 ]
        then
            continue
        fi
        printf "$j "
    done
done

```

```

loop 0:1 2
loop 1:1 2 3
loop 2:1 2 3
loop 3:1 2 3
loop 4:1 2 3 6
loop 5:1 2 3 6 7
loop 6:1 2 3 6 7 8
loop 7:1 2 3 6 7 8 9
loop 8:1 2 3 6 7 8 9 10

```

Continue will skip rest of the commands in the loop and goes to next iteration

`shift`

```

i=1
while [ -n "$1" ]
do
    echo argument $i is $1
    shift
    (( i++ ))
done

```

shift will shift the command line arguments by one to the left.

It keeps on shifting (read: delete/destroy) the arguments to the left

```

#!/bin/bash

echo "Number of args: ";
echo $#;
i=1;

while [ -n "$1" ]; do
    echo "arg $i is $1";
    shift;
    (( i++ ));
done;

echo "Number of args now: ";
echo $#;

```

exec

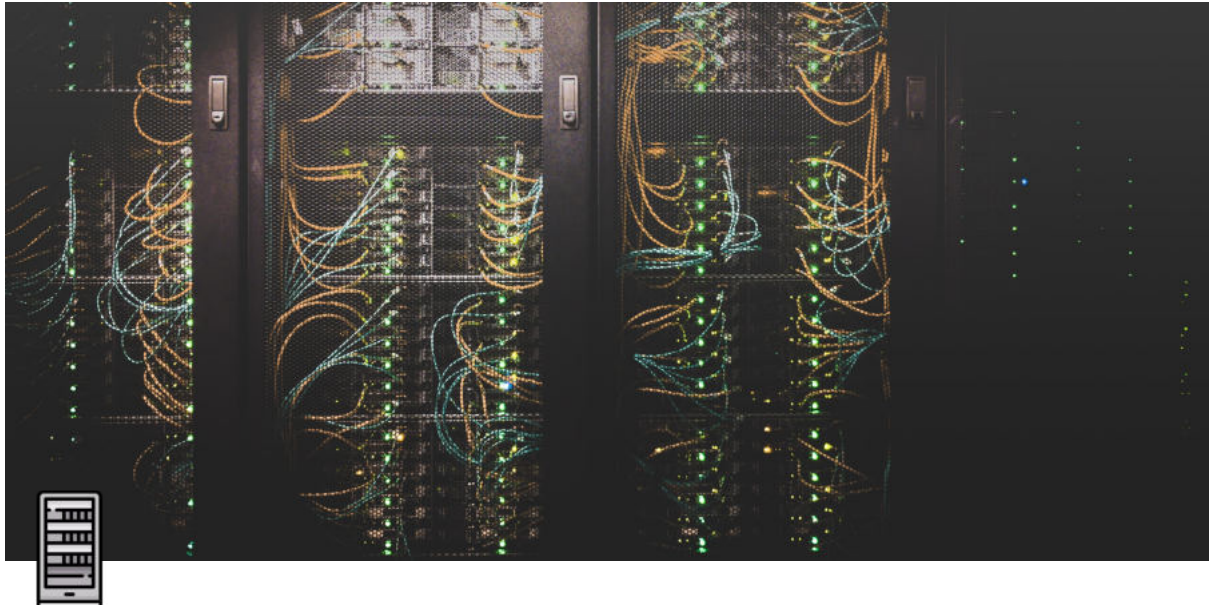
```
exec ./my-executable --my-options --my-args
```

- To replace shell with a new program or to change i/o settings
- If new program is launched successfully, it will not return control back to the shell
- If new program fails to launch, the shell continues


```

#!/bin/bash
echo "PID of shell running this command: $$";
echo "Leaving bash and opening xterm if available";
exec xterm;
echo "Looks like xterm is not available or failed to start";

```



Bash scripts - pt. 2C

▼ Type	 Lecture
📅 Date	@February 9, 2022
☰ Lecture #	3
🔗 Lecture URL	https://youtu.be/kjMhi4gI0po
🔗 Notion URL	https://21f1003586.notion.site/Bash-scripts-pt-2C-22f3a0910ba04a088ce5281aab4462e4
# Week #	6

eval

`eval my-arg`

- Executes argument as a shell command
- Combines arguments into a single string
- Returns control to the shell with exit status

```
#!/bin/bash
cmd="date";
```

```
fmt="+%d-%B-%Y";
eval $cmd $fmt;
```

Functions in bash

```
#!/bin/bash

usage() {
    echo "usage $1 str1 str2";
}

swap() {
    echo "$2 $1";
}

if [ $# -lt 2 ]; then
    usage $0;
    exit 1;
fi;

swap $1 $2;
```

getopts

```
while getopts "ab:c:" options;
do
    case "${options}" in
        b)
            barg=${OPTARG}
            echo accepted: -b $barg
            ;;
        c)
            carg=${OPTARG}
            echo accepted: -c $carg
            ;;
        a)
            echo accepted: -a
            ;;
        *)
            echo Usage: -a -b barg -c carg
            ;;
    esac
done
```

This script can be invoked with only three options: [a](#), [b](#), [c](#). The options [b](#) and [c](#) will take arguments.

```
#!/bin/bash
while getopts "ab:c:" options; do
    case "${options}" in
        b)
            barg=${OPTARG};
            echo "accepted: -b $barg";
            ;;
```

```

c)
    carg=${OPTARG};
    echo "accepted: -c $carg";
    ;;
a)
    echo "accepted: -b";
    ;;
*)
    echo "Usage: -a -b barg -c carg";
    ;;
esac;
done;

```

Usage

```

khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec1
$ bash getopts.sh -a -b bargument -c cargument
accepted: -b
accepted: -b bargument
accepted: -c cargument

khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec1
$ bash getopts.sh -a -b bargument -c cargument -d -e
accepted: -b
accepted: -b bargument
accepted: -c cargument
getopts.sh: illegal option -- d
Usage: -a -b barg -c carg
getopts.sh: illegal option -- e
Usage: -a -b barg -c carg

```

select loop

```

echo select a middle one
select i in {1..10}
do
    case $i in
        1 | 2 | 3)
            echo you picked a small one;;
        8 | 9 | 10)
            echo you picked a big one;;
        4 | 5 | 6 | 7)
            echo you picked the right one
            break;;
    esac
done
echo selection completed with $i

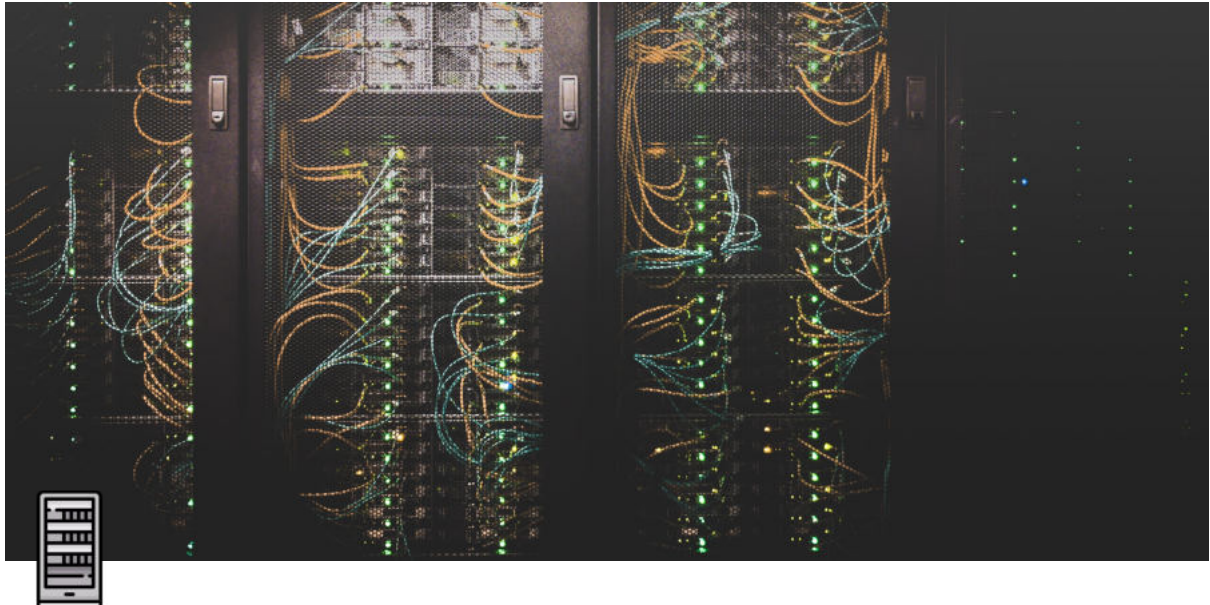
```

Text menu !



```
#!/bin/bash

echo "select a middle one";
select i in {1..10}; do
    case $i in
        1 | 2 | 3)
            echo "you picked a small one";
            ;;
        8 | 9 | 10)
            echo "you picked a big one";
            ;;
        4 | 5 | 6 | 7)
            echo "you picked the right one";
            break;
            ;;
    esac;
done;

echo "selection completed with $i";
```



awk programming - pt. 1

▼ Type	 Lecture
📅 Date	@February 9, 2022
☰ Lecture #	4
🔗 Lecture URL	https://youtu.be/k3q-9ntZl4s
🔗 Notion URL	https://21f1003586.notion.site/awk-programming-pt-1-63f859576b6c4836a7cc2ed96ac4c99b
# Week #	6

awk

A language for processing fields and records

Introduction

- It is a programming language
- **awk** is an abbreviation of the 3 people who developed it
 - **A**ho
 - **W**einberger

- Kernighan
- It is a part of POSIX, IEEE 1003.1-2008
- Variants of `awk`
 - `nawk`
 - `gawk`
 - `mawk`
 - ...
- `gawk` contains features that extend POSIX

Execution model

- Input stream is a set of records
 - Eg. using `"\n"` as a record separator, lines are recorded
- Each record is a sequence of fields
 - Eg. using `" "` as a field separator, words are fields
- Splitting of records to fields is done automatically
- Each codeblock executes on one record at a time, as matched by the pattern of that block

usage

- Single line at the command line

```
cat /etc/passwd | awk -F":" '{print $1}'
```

- Script interpreted by awk

```
./myscript.awk /etc/passwd
```

```
myscript.awk
```



```
#!/usr/bin/gawk -f
BEGIN {
    FS=":"
}
{
    print $1
}
```

Example #1

Block example

It does one print statement to illustrate how many times each codeblock is being processed

```
#!/usr/bin/gawk -f
BEGIN {
    print "BEGIN action is processed";
}
{
    print "Default action is processed";
}
END {
    print "END action is processed";
}
```

```
line-1 word1a word1b word1b
line-2 word2a word2b word2b
line-3 word3a word3b word3c
```

Output

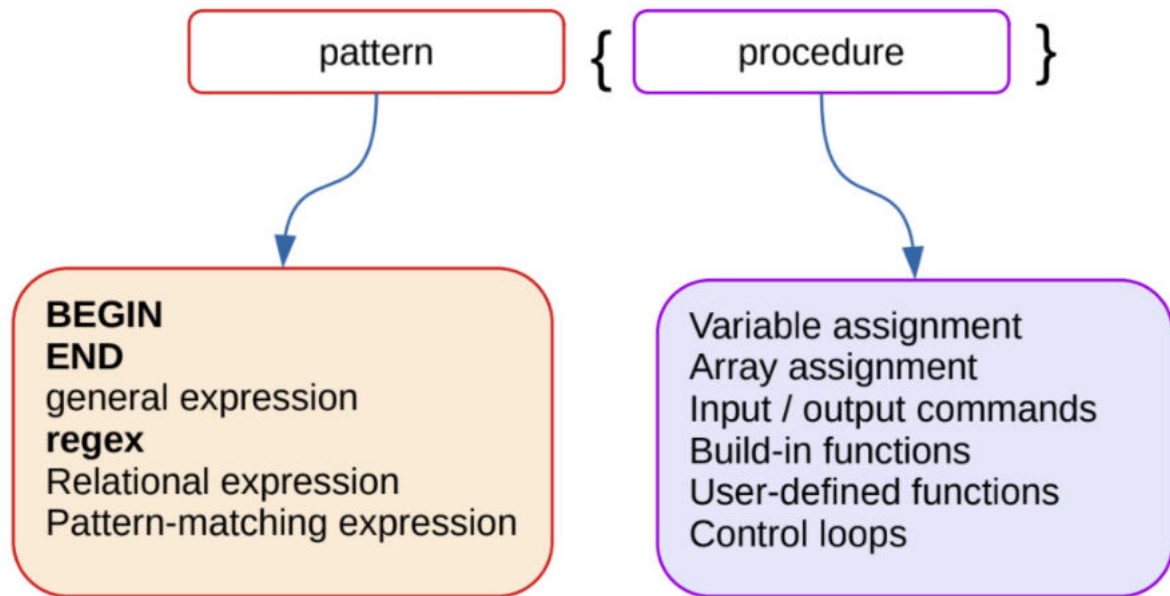
```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ ./block-ex1.awk block-ex1.input
BEGIN action is processed
Default action is processed
Default action is processed
Default action is processed
END action is processed

khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ cat block-ex1.input | ./block-ex1.awk
BEGIN action is processed
Default action is processed
Default action is processed
Default action is processed
END action is processed
```

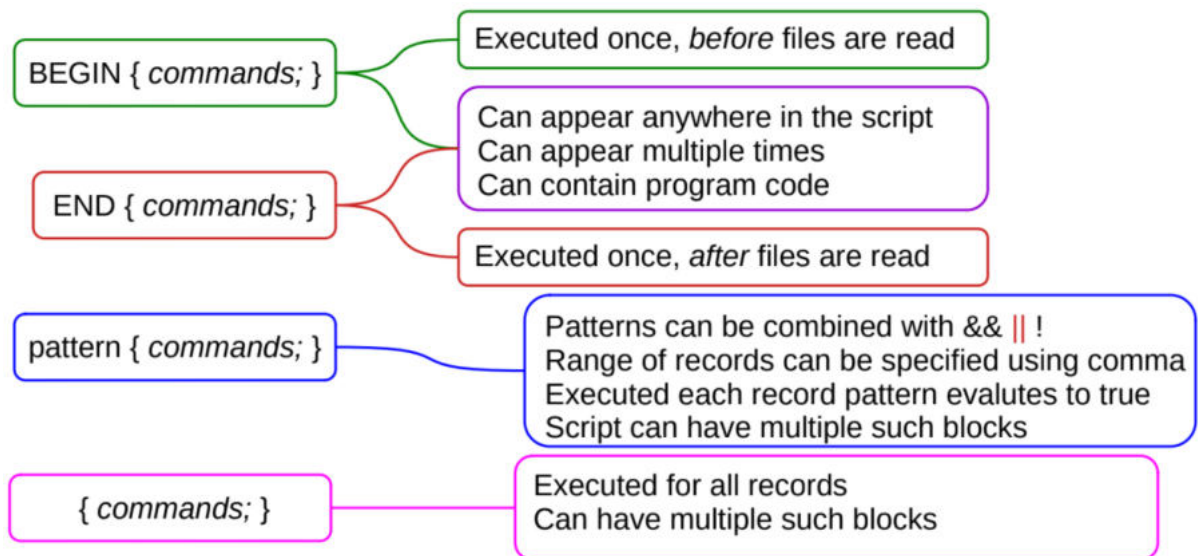
Built-in variables

ARGC	Number of arguments supplied on the command line (except those that came with -f & -v options)
ARGV	Array of command line arguments supplied; indexed from 0 to ARGC-1
ENVIRON	Associative array of environment variables
FILENAME	Current filename being processed
FNR	Number of the current record, relative to the current file
FS	Field separator, can use regex
NF	Number of fields in the current record
NR	Number of the current record
OFMT	Output format for numbers
OFS	Output fields separator
ORS	Output record separator
RS	Record separator
RLENGTH	Length of string matched by match() function
RSTART	First position in the string matched by match() function
SUBSEP	Separator character for array subscripts
\$0	Entire input record
\$n	n th field in the current record

awk scripts



execution



operators

Assignment	= += -= *= /= %= ^= **=
Logical	&&
Algebraic	+ - * / % ^ **
Relational	> <= > >= != ==

<code>expr ? a : b</code>	Conditional expression
<code>a in array</code>	Array membership
<code>a ~ /regex/</code>	Regular expression match
<code>a !~ /regex/</code>	Negation of regular expression match
<code>++</code>	Increment, both prefix and postfix
<code>--</code>	decrement, both prefix and postfix
<code>\$</code>	Field reference
	Blank is for concatenation

Functions and Commands

Arithmetic	<code>atan2</code> <code>cos</code> <code>exp</code> <code>int</code> <code>log</code> <code>rand</code> <code>sin</code> <code>sqrt</code> <code>srand</code>
String	<code>asort</code> <code>asorti</code> <code>gsub</code> <code>index</code> <code>length</code> <code>match</code> <code>split</code> <code>sprintf</code> <code>strtonum</code> <code>sub</code> <code>substr</code> <code>tolower</code> <code>toupper</code>
Control Flow	<code>break</code> <code>continue</code> <code>do</code> <code>while</code> <code>exit</code> <code>for</code> <code>if</code> <code>else</code> <code>return</code>
Input / Output	<code>close</code> <code>fflush</code> <code>getline</code> <code>next</code> <code>nextline</code> <code>print</code> <code>printf</code>
Programming	<code>extension</code> <code>delete</code> <code>function</code> <code>system</code>
bit-wise	<code>and</code> <code>compl</code> <code>lshift</code> <code>or</code> <code>rshift</code> <code>xor</code>

```
#!/usr/bin/gawk -f
BEGIN {
    print "BEGIN action is processed";
}
{
    print "record: " $0;
    print "processing record number: " FNR;
    print "number of fields in the current record: " NF;
```



```

}
END {
    print "END action is processed";
}

```

```

line-1 word1a word1b word1b
line-2 word2a word2b
line-3 word3a

```

```

khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ ./block-ex2.awk block-ex2.input
BEGIN action is processed
record: line-1 word1a word1b word1b
processing record number: 1
number of fields in the current record: 4
record: line-2 word2a word2b
processing record number: 2
number of fields in the current record: 3
record: line-3 word3a
processing record number: 3
number of fields in the current record: 2
END action is processed

```

```

#!/usr/bin/gawk -f
BEGIN {
    print "BEGIN action is processed";
}
/[[alpha:]]/ {
    print "alpha record: " FNR ":" $0;
    print "number of fields in the current record: " NF;
}
/[[alnum:]]/ {
    print "alnum record: " FNR ":" $0;
    print "number of fields in the current record: " NF;
}
/[[digit:]]/ {
    print "digit record: " FNR ":" $0;
    print "number of fields in the current record: " NF;
}
END {
    print "END action is processed";
}

```

```

hello world welcome to awk
mm22b901 name place
600036 mm23b902 name
04422574770 231

```


Output

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ ./block-ex3.awk block-ex3.input
BEGIN action is processed
alpha record: 1:hello world welcome to awk
number of fields in the current record: 5
alnum record: 1:hello world welcome to awk
number of fields in the current record: 5
alpha record: 2:mm22b901 name place
number of fields in the current record: 3
alnum record: 2:mm22b901 name place
number of fields in the current record: 3
digit record: 2:mm22b901 name place
number of fields in the current record: 3
alpha record: 3:600036 mm23b902 name
number of fields in the current record: 3
alnum record: 3:600036 mm23b902 name
number of fields in the current record: 3
digit record: 3:600036 mm23b902 name
number of fields in the current record: 3
alnum record: 4:04422574770 231
number of fields in the current record: 2
digit record: 4:04422574770 231
number of fields in the current record: 2
END action is processed
```

To match only the first field, instead of the entire record

```
#!/usr/bin/gawk -f
BEGIN {
    print "BEGIN action is processed";
}
$1 ~ /^[[:alpha:]]/ {
    print "alpha record: " FNR ":" $0;
    print "number of fields in the current record: " NF;
}
$1 ~ /^[[:alnum:]]/ {
    print "alnum record: " FNR ":" $0;
    print "number of fields in the current record: " NF;
}
$1 ~ /^[[:digit:]]/ {
    print "digit record: " FNR ":" $0;
    print "number of fields in the current record: " NF;
}
END {
    print "END action is processed";
}
```

Input is the same as the previous

Output

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ ./block-ex4.awk block-ex3.input
BEGIN action is processed
alpha record: 1:hello world welcome to awk
number of fields in the current record: 5
alnum record: 1:hello world welcome to awk
number of fields in the current record: 5
alpha record: 2:mm22b901 name place
number of fields in the current record: 3
alnum record: 2:mm22b901 name place
number of fields in the current record: 3
digit record: 2:mm22b901 name place
number of fields in the current record: 3
alnum record: 3:600036 mm23b902 name
number of fields in the current record: 3
digit record: 3:600036 mm23b902 name
number of fields in the current record: 3
alnum record: 4:04422574770 231
number of fields in the current record: 2
digit record: 4:04422574770 231
number of fields in the current record: 2
END action is processed
```

In the following example, we don't do any pattern matching or comparison

Instead, we look at the number of fields in any particular (or arbitrary) record

```
#!/usr/bin/gawk -f
BEGIN {
    print "BEGIN action is processed";
    FS="[ .;:-]";
}
NF > 2 {
    print "acceptable record:" FNR ":" $0;
    print "number of fields in the current record:" NF;
}
NF <= 2 {
    print "not acceptable record:" FNR ":" $0;
    print "number of fields in the current record:" NF;
}
END {
    print "END action is processed";
}
```

Input

```
hello:world:welcome to awk
mm22b901;name;place
600036-mm23b902-name
04422574770 231 12345
gphani@iitm.ac.in
```

Output

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ ./block-ex5.awk block-ex5.input
BEGIN action is processed
acceptable record:1:hello:world:welcome to awk
number of fields in the current record:5
acceptable record:2:mm22b901;name;place
number of fields in the current record:3
acceptable record:3:600036-mm23b902-name
number of fields in the current record:3
acceptable record:4:04422574770 231 12345
number of fields in the current record:3
acceptable record:5:gphani@iitm.ac.in
number of fields in the current record:3
END action is processed
```

We can modify the FS (field separator) in the above script to not allow " " (space) and "." (dot) as the field separator

The result will change

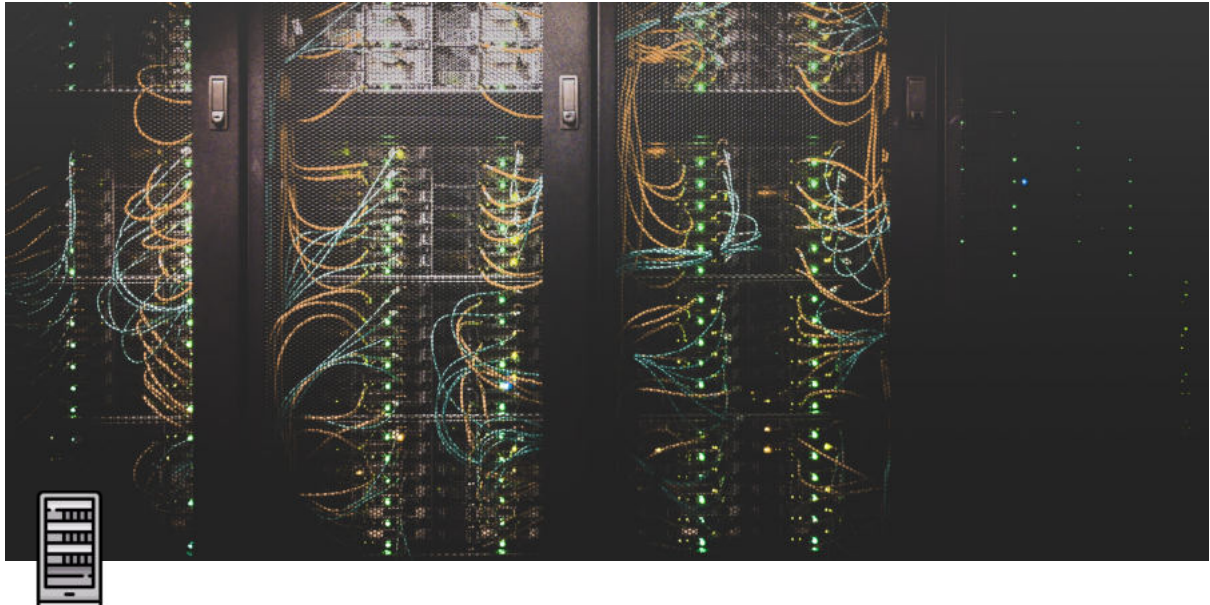
Modified code (Notice line #4)

```
BEGIN {
    print "BEGIN action is processed";
    FS="[:;-]";
}
NF > 2 {
    print "acceptable record:" FNR ":" $0;
    print "number of fields in the current record:" NF;
}
NF <= 2 {
    print "not acceptable record:" FNR ":" $0;
    print "number of fields in the current record:" NF;
}
END {
    print "END action is processed";
}
```


Input is the same as previous unmodified one

Output

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ ./block-ex5.awk block-ex5.input
BEGIN action is processed
acceptable record:1:hello:world:welcome to awk
number of fields in the current record:3
acceptable record:2:mm22b901;name;place
number of fields in the current record:3
acceptable record:3:600036-mm23b902-name
number of fields in the current record:3
not acceptable record:4:04422574770 231 12345
number of fields in the current record:1
not acceptable record:5:gphani@iitm.ac.in
number of fields in the current record:1
END action is processed
```



awk programming - pt. 2

▼ Type	 Lecture
📅 Date	@February 9, 2022
☰ Lecture #	5
🔗 Lecture URL	https://youtu.be/g9_ij64u8eQ
🔗 Notion URL	https://21f1003586.notion.site/awk-programming-pt-2-b59debe0a4b54507a20eadf4bc0e1042
# Week #	6

Arrays

- Associative arrays
- Sparse storage
- Index need not be integer
- `arr[index]=value`
- `for (var in arr)`
- `delete arr[index]`

Loops

```
for (a in array)
{
    print a
}
```

```
if (a > b)
{
    print a
}
```

```
for (i=1;i<n;i++)
{
    print i
}
```

```
while (a < n)
{
    print a
}
```

```
do
{
    print a
} while (a < n)
```

awk program to calculate the total fee, avg fee and the roll numbers (and their fee) of those who paid less than a threshold from a garbled data file

block-ex6.awk

```
#!/usr/bin/gawk -f
BEGIN {
    print "Report of fee paid:";
    totfee=0;
    FS=" ";
}
{
    # print $0;
    if ($1 ~ /^[[:alpha:]]{2}[[:digit:]]{2}[[:alpha:]]{1}[[:digit:]]{3}+$/) {
        roll=$1;
        fee=$2;
        rf[roll]=fee;
        totfee += fee;
        print roll " paid " fee;
    }
}
END {
    cutoff=21500;
    print "List of students who paid less than " cutoff;
    ns=0;
    for (r in rf)
    {
        ns++;
        if(rf[r] < cutoff) print "check ", r, " paid only " rf[r];
    }
    avg = totfee/ns;
    print "Total fee collected:" totfee;
    print "Average fee collected:" avg;
}
```


Input

block-ex6.input

```
This file contains comments and numbers
Lines containing roll number and the fee paid need to be picked up
mm22b901 21000
Sum of the total fee paid needs to be printed at the end
Also a list of students, who paid how much
mm22b902 21000
Input text may contain some comments in between
mm22b906      21800
mm22b903 21500 paid through DD
mm22b905      22000
updated as on Jan 26, 2022 by Phani
```

Output

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ ./block-ex6.awk block-ex6.input
Report of fee paid:
mm22b901 paid 21000
mm22b902 paid 21000
mm22b906 paid 21800
mm22b903 paid 21500
mm22b905 paid 22000
List of students who paid less than 21500
check mm22b901 paid only 21000
check mm22b902 paid only 21000
Total fee collected:107300
Average fee collected:21460
```

functions

```
cat infile |awk -f mylib -f myscript.awk
```

mylib

```
function myfunc1()
{
    printf "%s\n", $1
}

function myfunc2(a)
{
    return a*rand()
}
```

myscript.awk

```
BEGIN
{
    a=1
}
{
    myfunc1()
    b = myfunc2(a)
    print b
}
```

func-lib.awk

```
function myfunc1() {
    printf "%f\n", 2*$1;
}
function myfunc2(a) {
    # Adding zero to 'a' here just to store the value in 'b' as an integer
    b=a+0;
    return sin(b);
}
```

func-example.awk

```
BEGIN {
    FS=".";
    print "----- BEGIN -----";
    c=atan2(1,1);
    print "c=" c;
    print "-----";
}
{
    print $0;
    myfunc1();
    a=$1;
    b=myfunc2(a);
    print "b=" b;
}
END {
    print "----- ADD -----";
    d=myfunc2(c);
    print "d=" d;
    print "-----";
}
```

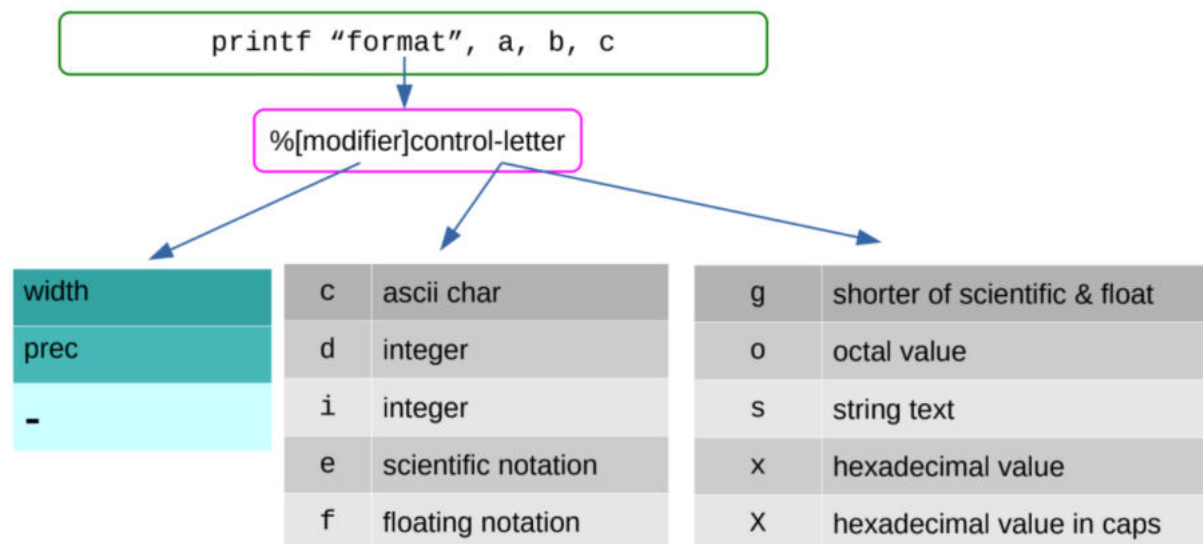

Output

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ echo "12.5" | gawk -f func-lib.awk -f func-example.awk
----- BEGIN -----
c=0.785398
-----
12.5
25.000000
b=-0.0663219
----- ADD -----
d=0.707107
-----
```

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ echo "" | gawk -f func-lib.awk -f func-example.awk
----- BEGIN -----
c=0.785398
-----
0.000000
b=0
----- ADD -----
d=0.707107
-----
```

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ touch xaa; cat xaa | gawk -f func-lib.awk -f func-example.awk
----- BEGIN -----
c=0.785398
-----
----- ADD -----
d=0.707107
-----
```

Pretty printing



bash + **awk**

- Including awk inside shell script
- heredoc feature
- Use with other shell scripts on command line using pipe

Examples

The following code creates 3 columns of 10 rows filled with random numbers

We also need to pass an input string for this code

But as we do not have a body for the action block, empty string suffices

rsheet-create.awk

```
#!/usr/bin/gawk -f
BEGIN {
    nl = 10;
    nc = 3;
    for(j=0; j<nl; j++) {
        for(i=0; i<nc; i++) {
            printf("%f ", rand());
        }
        printf("\n");
    }
}
{}
END {
    print nl " lines with " nc " columns of random numbers created";
}
```

Output

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ echo " " | ./rsheet-create.awk
0.924046 0.593909 0.306394
0.578941 0.740133 0.786926
0.436370 0.332195 0.778880
0.100887 0.785084 0.835159
0.761209 0.496077 0.426298
0.945798 0.821802 0.709269
0.157828 0.119752 0.909685
0.868084 0.449256 0.705432
0.399686 0.645049 0.696163
0.300211 0.591664 0.956569
10 lines with 3 columns of random numbers created
```

Now, we can modify the above code to generate 2 columns of 2 million rows filled with random numbers

We will pass the empty file `xaa` we created earlier, and also use the command `time` to calculate the time taken to execute

```
#!/usr/bin/gawk -f
BEGIN {
    nl = 2000000;
    nc = 2;
    for(j=0; j<nl; j++) {
        for(i=0; i<nc; i++) {
            printf("%f ", rand());
        }
        printf("\n");
    }
}
{}
END {
    # print nl " lines with " nc " columns of random numbers created";
}
```

Output

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ time ./rsheet-create.awk xaa > rsheet-data.txt

real    0m17.544s
user    0m17.500s
sys     0m0.078s
```

Well, it took a long time on my 🍌 laptop

Now, we should process these numbers and create 2 more rows after that which would contain the product and the sum of these numbers

`rsheet-process.awk`

```
#!/usr/bin/gawk -f
BEGIN {
    OFS=" ";
    FS=" ";
}
{
    prod = $1*$2;
    sum = $1+$2;
    printf("%f %f %f %f\n", $1, $2, prod, sum);
}
END {}
```

Input is the `rsheet-data.txt` file generated earlier

Output

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ time ./rsheet-process.awk rsheet-data.txt > rsheet-pdata.txt

real    0m53.066s
user    0m52.671s
sys     0m0.077s
```

An awk program to know the IP addresses of all the clients that connected to our web server (Apache) in the last 5 days

Server log file [76.6MB]: <https://firebasestorage.googleapis.com/v0/b/fb-sandbox-25.appspot.com/o/access-full.log?alt=media&token=b1e22c89-5f85-4e12-8087-dfe50fa0e49d>

To get the IP addresses

Type the following code on the bash terminal

```
awk 'BEGIN {FS=" "}{print $1}' access-head.log
```

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ awk 'BEGIN {FS=" "}{print $1}' access-head.log
103.47.219.249
54.209.123.136
54.209.123.136
54.209.123.136
54.209.123.136
54.209.123.136
54.209.123.136
54.209.123.136
54.209.123.136
54.209.123.136
```

```
awk 'BEGIN {FS=" "}{print $1}' access-tail.log
```

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ awk 'BEGIN {FS=" "}{print $1}' access-tail.log
52.203.104.35
52.203.104.35
52.203.104.35
52.203.104.35
52.203.104.35
52.203.104.35
52.203.104.35
52.203.104.35
52.203.104.35
52.203.104.35
```

Here, `access-head.log` and `access-tail.log` are the top 10 and bottom 10 lines of the file `access-full.log` respectively

Similarly, to get the date

Type the following code on the bash terminal

```
awk 'BEGIN {FS=" "}{d=substr($4,2,11); print d}' access-head.log
```

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ awk 'BEGIN {FS=" "}{d=substr($4,2,11); print d}' access-head.log
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
```

```
awk 'BEGIN {FS=" "}{d=substr($4,2,11); print d}' access-tail.log
```

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ awk 'BEGIN {FS=" "}{d=substr($4,2,11); print d}' access-tail.log
14/Jan/2022
14/Jan/2022
14/Jan/2022
14/Jan/2022
14/Jan/2022
14/Jan/2022
14/Jan/2022
14/Jan/2022
14/Jan/2022
14/Jan/2022
```

Now, combine the above 2 scripts (one from the IP address one, one from the date one) to get the IP address and the date on which they connected to the server

```
awk 'BEGIN {FS=" "}{d=substr($4,2,11); print d, $1}' access-head.log
```

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ awk 'BEGIN {FS=" "}{d=substr($4,2,11); print d, $1}' access-head.log
27/Jan/2022 103.47.219.249
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
```

```
awk 'BEGIN {FS=" "}{d=substr($4,2,11); print d, $1}' access-tail.log
```

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ awk 'BEGIN {FS=" "}{d=substr($4,2,11); print d, $1}' access-tail.log
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
```

Now, run it on the entire log file and `time` it too. Also save it to a file named `date-ip.txt`

```
time awk 'BEGIN {FS=" "}{d=substr($4,2,11); print d, $1}' access-full.log > date-ip.txt
```

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ time awk 'BEGIN {FS=" "}{d=substr($4,2,11); print d, $1}' access-full.log > date-ip.txt

real    0m0.688s
user    0m0.625s
sys     0m0.031s
```

Statistics time 😊

To calculate the number of connections per day from the log file

To get the datestring for the last n days


```
date --date="<n> days ago" +%d/%m/%Y
```

NOTE: Replace the `<n>` with the actual number

Example

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ date --date="5 days ago" +%d/%m/%Y
05/02/2022

khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ date --date="2 days ago" +%d/%m/%Y
08/02/2022

khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk
$ date --date="3 days ago" +%d/%m/%Y
07/02/2022
```

Code

apache-log-ex1.awk

```
#!/usr/bin/gawk -f
BEGIN {
    ndays=15;
    dformat="+%d/%b/%Y";
    for(i=0; i<ndays; i++) {
        cmdstr=sprintf("date --date=\"%d days ago\" %s", i, dformat);
        cmdstr | getline mydate;
        dates[i]=mydate;
    }
    dstring = "";
    for (i in dates) {
        dstring = dstring " " dates[i];
    }
    print "date string=" dstring;
}
{
    ldate=substr($4,2,11);
    w = match(dstring,ldate);
    if(w != 0) {
        # print ldate " " $1 " " $7;
        print ldate " " $1;
        ipcount[$1]++;
    }
}
END {
    print "----- IP STATS -----";
    for (j in ipcount) {
        print j " " ipcount[j];
    }
}
```



```
}  
}
```

Input

`access-head.log` or `access-tail.log` or `access-full.log`

Output

```
khaqu@DESKTOP-77CS341 MINGW64 ~/Documents/iitm-term4/sc/week6/lec-awk  
$ ./apache-log-ex1.awk access-head.log  
date string= 10/Feb/2022 09/Feb/2022 08/Feb/2022 07/Feb/2022 06/Feb/2022  
8/Jan/2022 27/Jan/2022  
27/Jan/2022 103.47.219.249  
27/Jan/2022 54.209.123.136  
27/Jan/2022 54.209.123.136  
27/Jan/2022 54.209.123.136  
27/Jan/2022 54.209.123.136  
27/Jan/2022 54.209.123.136  
27/Jan/2022 54.209.123.136  
27/Jan/2022 54.209.123.136  
27/Jan/2022 54.209.123.136  
27/Jan/2022 54.209.123.136  
----- IP STATS -----  
103.47.219.249 1  
54.209.123.136 9
```

DNS lookup utility

Command: `dig`

Usage: `dig -x <ip-address>`

Example

```
kashif@zen:~$ dig -x 54.209.123.136  
  
; <<> DiG 9.16.1-Ubuntu <<> -x 54.209.123.136  
;; global options: +cmd  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46107  
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1  
  
;; OPT PSEUDOSECTION:  
; EDNS: version: 0, flags:; udp: 65494  
;; QUESTION SECTION:  
;136.123.209.54.in-addr.arpa. IN PTR  
  
;; ANSWER SECTION:  
136.123.209.54.in-addr.arpa. 300 IN PTR ec2-54-209-123-136.compute-1.amazonaws.com.  
  
;; Query time: 88 msec  
;; SERVER: 127.0.0.53#53(127.0.0.53)  
;; WHEN: Fri Feb 11 09:08:39 UTC 2022  
;; MSG SIZE rcvd: 112
```

To get a slimmer output: `dig +noall +answer -x 34.234.167.93`

Example

```
kashif@zen:~$ dig +noall +answer -x 34.234.167.93
93.167.234.34.in-addr.arpa. 300 IN PTR ec2-34-234-167-93.compute-1.amazonaws.com.
```

Previous `awk` code, modified

```
#!/usr/bin/gawk -f
BEGIN {
    ndays=25;
    dformat="+%d/%b/%Y";
    for(i=0; i<ndays; i++) {
        cmdstr=sprintf("date --date=\"%d days ago\" %s", i, dformat);
        cmdstr | getline mydate;
        dates[i]=mydate;
    }
    dstring = "";
    for (i in dates) {
        dstring = dstring " " dates[i];
    }
    print "date string=" dstring;
}
{
    ldate=substr($4,2,11);
    w = match(dstring,ldate);
    if(w != 0) {
        # print ldate " " $1 " " $7;
        # print ldate " " $1;
        ipcount[$1]++;
    }
}
END {
    print "----- IP STATS -----";
    for (j in ipcount) {
        print j " " ipcount[j];
        cmdstr = sprintf("dig +noall +answer -x %s", j);
        cmdstr | getline ipinfo;
        print ipinfo;
    }
}
```