# System Administration

BOOSTER
boost your career

## Session 6 CONTENT

- sed utility
- awk utility
- Why Shell Scripting?
- Using / Avoid using Shell Script
- Variables in linux
- Flow Control

## What is sed?

- It is a streamline, **non-interactive** editor.
- It performs the same kind of tasks as in vi.
- It doesn’t change your file unless the output is saved with shell redirection by default.
- The sed editor process a file (input) one line at a time and sends its output to the screen.

- The sed stores the line it process in a buffer and once processing is finished the line is sent to the screen (unless command was delete) and the next line is read and the process is repeated until the last line is reached.

# Addressing

- Addressing is used to determine which lines to be edited.
- The addressing format can be

  Number (represents a line number)

  Regular expression

  Both

- The sed commands tell sed what to do with the line:
  - Print it
  - Remove it
  - Change it
- The sed format

```
sed 'command' filename
```

## Examples

- To print lines contain the pattern root

```
$sed '/root/p' myfile
```

- To suppresses the default behavior of the  sed

```
$sed -n '/root/p' myfile
```

- To print lines from mohammed to root

```
$sed -n '/mohammed/,/root/p' myfile
```

- To print lines from 2 to the line that begins with us

```
$sed -n '2,/^us/p' myfile
```

## Examples

- To delete lines from 1 to 3 (including 3rd line)

```
$sed ‘1,3d’ myfile
```

- To delete from line 3 to the end

```
$sed ‘3,$d’ myfile
```

- To delete lines containing root pattern

```
$sed ‘/root/d’ myfile
```

- To delete the third line

```
$sed ‘3d’ myfile
```

- To delete the last line

```
$sed ‘$d’ myfile
```

- To substitute text

```
$sed ‘s/mohammed/ahmed/g’ myfile
$sed –n ‘s/mohammed/ahmed/gp’ myfile
```

## Quiz

How can you make multiple changes on different lines using sed?

## The AWK Utility

- What is AWK?
- What does AWK stands for?
- The awk’s format
- Records and Fields
- Examples
- BEGIN Pattern
- END Pattern
- Conditional Expressions
- Loops
- Examples

## What is AWK?

- awk is a programming language used for manipulating data and generating reports.
- awk scans a file line by line, searching for lines that match a specified pattern  performing selected actions
- awk stands for the first initials in the last names of each authors of the language, Alfred Aho, Peter Weinberger, and Brian Kernighan

## awk Format

- The awk utility consists of
  - awk command
  - Program instructions enclosed in quotes
  - Input file or default stdin.

```
$awk ‘instructions’ inputfile
```

## Records and fields

By default, each line is called a record and terminated with a new line.

- Record separators are by default carriage return, stored in a built-in variables `ORS` and `RS`.
- The `$0` variable is the entire record.
- The `NR` variable is the record number.

Each record consists of words called fields which by default separated by white spaces.

- `NF` variables contains the number of fields in a record
- `FS` variable holds the input field separator, space/tab is the default.

## Examples

- To print the first field of the file, but as the default delimiter is white space, you have to specify the delimiter

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

$awk -F: '{print "Logname:",$1}' /etc/passwd
```

## Examples

- To display the whole file (cat)

```
$awk '{print $0}' /etc/passwd
```

- To display the file numbered (cat -n)

```
$awk '{print NR,$0}' /etc/passwd
```

- To display number of fields (words) in each record (line)

```
$awk -F: '{print $0,NF}' /etc/passwd
```

## BEGIN Pattern

- BEGIN Pattern is followed by an action block that is executed before awk process any line from the input file.

- BEGIN action is often used to change the value of the built-in variables, FS, RS, and so forth to assign initial values to user-defined variables and print headers of titles.

Example

```
$awk 'BEGIN{FS=":"; RS="\n\n"} {print $1,$2,$3}' myfile
```

# END Pattern

- END patterns are handled after all lines of input have been processed.
- It does not match any input line
- Example:

  To print the number of lines in a file

```
$awk ‘END { print NR } ’ testfile
```

## Conditional expressions

```
if (expression1){
statement; statement;...
}
else if (expression2){
statement; statement;...
}
else {
statement
}
```

## Relational Operators

| Operator | Meaning |
|---|---|
| < | Less than |
| <= | Less than and equal |
| == | Equal to |
| != | Not equal to |
| >= | Greater than and equal |
| > | Greater than |
| ~ | Match regular expression |
| !~ | Not matched by regular expression |

## Loops

- while Loop

```
$awk -F: '{i=1; while (i<NF) {print NF,$i;i++}}' /etc/passwd
```

- For Loop

```
$awk -F: '{for ( i=1 ; i<NF; i++) print NF,$i}' /etc/passwd
```

# Examples (cont.)

- The variable max value is set according to comparison between the first 2 fields:

```
$ awk '{if ($1>$2)
    max=$1;
else
    max=$2;
print max}' testing
```

- Arithmetical Operations

```
$ awk '{if ($1*$2>100) print $0}' testing
```

- To display line 4 and 5 only

```
$awk '{if (NR==4 || NR==5)print NR":"$0}' /etc/passwd
```

# Why Shell Scripting?

- Bash shel scripts:
  - Execute bash commands from a file
  - Automate sequences of shell commands
  - Run them regularly or at scheduled times
- It supports the user by allowing tools for
  - Data selection
  - Data combination
  - Decision and rules.
- Processing text
- File manipulation
- To automate repetitive tasks
- It's so simple
- It's portable

## When do you AVOID Shell script

- This task will be done once.
- It is a complex task and need user interactive.
- Doing complex mathematical calculations
- Handling binary data

# Standard Shells

- Bourne Shell (sh)
  - Most system Administration’s scripts are Bourne shell scripts as it’s the default shell in most linux distributions
- C Shell (csh)
  - Command line history, aliasing, and job control

- sh shell is more simpler and faster

- Korn shell (ksh)
  - Editing history, aliasing, functions, regular expression wildcards, job control and special debugging features

## Creating a shell Script

- A shell program is a combination of UNIX commands, programming constructions and comments.
- To execute the script use `chmod` command to turn on the **execute** permission.

The first line

```
#!/usr/bin/bash
```

Comments

```
# calculating x
```

## Running a shell Script

To be able to  run a shell script make sure of permissions

- Executing

```
./script
/home/nagger/script.sh
```

- Sourcing

```
. ./script.sh
source /home/nagger/script.sh
```

## Variables

- Type of Variables:
  - Local Variables
  - Environment Variables
  - Predefined Variables

# Local Variables

- Local variables are given values that are known only to the shell in which they are created.
- Variables names must begin with an alphabetic or underscore character.

- Example

```
$ fname=mohammed
$ echo $firstname
$ lname=alnaggar
$ echo $fname $lname
        mohammed alnaggar
```

## Environment Variables

Environment variables are available to the shell in which they are created and any sub-shells.

```
PATH
HOME
PS1
LOGNAME
PS2
….
```

## Environment Variables

- To set an environment variable

```
export VAR=value
```

- To unset a variable

```
unset VAR
```

- To display all variables

```
env command
```

- To display values stored in variables

```
echo $VAR
```

# Predefined Variables

Predefined variables are variables known to the shell and their values are assigned by the shell.

| | |
|---|---|
| • `$#` | Number of arguments |
| • `$*` | List of all arguments |
| • `$0` | Script name |
| • `$1, $2,..` | First argument, second argument,.. |
| • `$?` | Return code of the last command |

## Reading User Input

```
#!/usr/bin/ksh

echo “What is your full name?”
read name
print “hello $name”

print “where do you work?”
read
print I guess $REPLY keeps you busy !
```

# Conditional Constructs and Flow control

- Conditional commands allow you to perform some tasks based on whether or not a condition succeeds or fails
  - `if`
  - `if/else`
  - `if/elif/else`

- The shell expects a command after an `if` and the exit status of the command is used to evaluate the condition.

- To evaluate an expression use the `test` command or double brackets.

# Testing and logical operations

- String Testing

| | |
|---|---|
| `String1 = string2` | string1 is equal string2 |
| `String1 != string2` | string1 is not equal string2 |
| `-z string` | Length of string is zero |
| `-n string` | Length of string is nonzero |
| `-n string` | Length of string is nonzero |

- Integer Testing

| | |
|---|---|
| `int1 -eq int2` | equal to |
| `int1 -ne int2` | not equal to |
| `int1 -gt int2` | greater than |
| `int1 -ge int2` | greater than or equal |
| `int1 -lt int2` | less than |
| `int1 -le int2` | less than or equal |

## Testing and logical operations (cont.)

|  |  |
|---|---|
| `-a` | and operator |
| `-o` | or operator |
| `-f filename` | file existence |
| `-h filename` | symbolic link |
| `-r filename` | readable |
| `-w filename` | writable |
| `-x filename` | executable |

## Examples

```
$ test -r fname

$ test “islam” != “isla”

$ test 5 -gt 3

$ test “mohamed” = “mohammed” -a 5 -gt 3
```

## The `if` command

```
if command
then
    … commands …
fi
        or
if [ expression ]
then
    … commands …
fi

if command
then
    … commands …
    if command
    then
        … commands …
    fi
fi
```

## Example

```
$ if  test  -f   file1
  >then
  >cat file1
>fi

$ if  [  -f   file1  ]
  > then
  > cat file1
>fi

$ print “Are you ok?”
$ read answer
$ if [ $answer = Y –o $answer = y ]
    > then
   > print “Glad to hear that ☺”
➢ fi
```

BOOSTER boost your career

# Questions?!

# Thank YOU!