# OSL640: INTRODUCTION TO OPEN SOURCE SYSTEMS

## WEEK 11 LESSON 1

## THE SED UTILITY

PHOTOS AND ICONS USED IN THIS SLIDE SHOW ARE LICENSED UNDER CC BY-SA

## LESSON 1 TOPICS

**The `sed` Utility**

- Definition / Purpose
- Usage
- Using **`sed`** as a Filter with Pipeline Commands
- Demonstration

**Perform Week 11 Tutorial**

- Investigation 1
- Review Questions (**Parts A** and **B**)

## SED UTILITY

### Purpose

The **sed** command stands for **Streaming Editor**.

The sed command is used to **manipulate text** that is contained in a **text file** or via a **pipeline command**.

Although the sed command does NOT change content inside a text file, this command acts like a *"on-the-fly"* text editor to display modified text on the **screen**, **redirect** to a file or act as a **filter** within a pipeline command.

## SED UTILITY

**Usage:**

```
sed [-n] 'address instruction' filename
```

**How it Works:**

- The sed command reads **all lines in the input file** and will be exposed to the expression (i.e. area contained within **quotes**) one line at a time.
- The expression can be within **single** quotes or **double** quotes.
- The **expression** contains an **address** (match condition) and an **instruction** (operation).
- If the line matches the **address**, then it will perform the **instruction**.
- Lines will display be default unless the **–n** option is used to <u>suppress</u> default display

## SED UTILITY

**Usage:**

```
sed [-n] 'address instruction' filename
```

**Addresses:**

- Can use a **line number**, to select a specific line (for example: `5`)
- Can specify a **range of line numbers** (for example: `5,7`)
- Regular expressions are contained within **forward slashes** (e.g. `/regular-expression/`)
- Can specify a **regular expression** to select all lines that match a pattern (e.g `/^[0-9].*[0-9]$/`)
- If **NO** address is present, the **instruction** will apply to **ALL** lines

## SED UTILITY

**Usage:**

```
sed [-n] 'address instruction' filename
```

**Common Instructions:**

**p** **Print** lines that match the address (commonly used with **-n** option)

**d** Omit (**delete**) display of lines that match the address

**q** Print lines including line that matches address and then **quit** processing

**s** **Substitute** text to replace a matched regular expression (similar *search and replace*)

# SED UTILITY

## Example 1

The following sed command line displays all lines in the readme file that contain the word **line** (all lowercase).

In addition, because there is no **–n** option, sed displays all the lines of input.

As a result, sed displays the lines that contain the word line **twice**.

```
sed '/line/ p' readme
Line one.
The second line.
The second line.
The third.
This is line four.
This is line four.
Five.
This is the sixth sentence.
This is line 7.
This is line 7.
Eight and last.
```

Unless you instruct it not to, sed sends **all lines**, selected or not to standard output.

When you use the `–n` option on the command line, sed sends only those lines to stdout that you specify with the print `p` command

## SED UTILITY

**Example 2**

The following sed command displays contents of a file from a **range** of line numbers.

```
sed -n '3,6 p' readme
The third.
This is line four.
Five.
This is the sixth sentence.
```

The print `p` instruction using the `-n` option only displays lines **3** through **6**.

# SED UTILITY

## Example 3

The following sed command displays the first **five** lines of text just as **a head -5** lines command would.

```
sed '5 q' readme
Line one.
The second line.
The third.
This is line four.
Five.
```

The sed command prints **all lines**, beginning from the first line, In this example, sed will **terminate** when **line 5** is matched.

# SED UTILITY

## Example 4

The following sed command displays a **TAB** character for lines contained in a file.

```
$ sed 's/^./\t&/' readme
   Line one.
   The second line.
   The third.
   etc...
```

The regular expression in the following instruction (`^.`) matches one character at the beginning of every line that is not empty.

The replacement string (between the second and third forward slashes) contains a backslash escape sequence that represents a **TAB** character (`\t`) followed by an ampersand (`&`).

The **ampersand** character (`&`) takes on the value of what the regular expression matched.

# SED UTILITY

## Example 5

The following sed command uses a **regular expression** and the **quit** instruction.

```
sed '/[0-9][0-9][0-9]$/ q' myfile
sfun 11
cool 12
Super 12a
Happy112
```

The regular expression in the following expression `[0-9][0-9][0-9]$` matches **three digits** at the end of a line.

The command will process the file, one-line at a time, beginning at the top and (by default) outputting each line to standard output.

Once the regular expression is matched, it will display the matched line and stop processing the *sed* command.

# SED UTILITY

## Using sed Utility as a Filter with Pipeline Commands

Although sed can be used as a streaming editor for text contained within a text file, the sed command can also be used as a **filter** within a **pipeline command**.

### Examples

```
ls | sed 's/^[0-9]/x/g’
echo “I like Linux” | sed 's/ /,/g'
```

# SED UTILITY

## Instructor Demonstration

Your professor will demonstrate additional examples using the **sed** utility.

Pathname of cars database: `~osl640/cars.txt`

Commands

```
sed -n '3,6 p' cars.txt
sed '5 d' cars.txt
sed '5,8 d' cars.txt
sed '5 q' cars.txt
sed -n '/chevy/ p' cars.txt
sed '/chevy/ d' cars.txt
sed '/chevy/ q' cars.txt
sed 's/[0-9]/*/' cars.txt
sed 's/[0-9]/*/g' cars.txt
sed '5,8 s/[0-9]/*/' cars.txt
sed 's/[0-9][0-9]*/*** & ***/' cars.txt
```

Contents of **cars** database file:

```
plym fury 77 73 2500
chevy nova 79 60 3000
ford mustang 65 45 17000
volvo gl 78 102 9850
ford ltd 83 15 10500
Chevy nova 80 50 3500
fiat 600 65 115 450
honda accord 81 30 6000
ford thundbd 84 10 17000
toyota tercel 82 180 750
chevy impala 65 85 1550
ford bronco 83 25 9525
```

## SED UTILITY

### Getting Practice

To get practice perform **Week 11 Tutorial:**

- INVESTIGATION 1: USING THE SED UTILITY

- LINUX PRACTICE QUESTIONS (Parts A and B)

# OSL640: INTRODUCTION TO OPEN SOURCE SYSTEMS

## WEEK 11: LESSON 2

## THE AWK UTILITY

PHOTOS AND ICONS USED IN THIS SLIDE SHOW ARE LICENSED UNDER CC BY-SA

## LESSON 2 TOPICS

**The `awk` Utility**

- Definition / Purpose
- Usage
- Using **`awk`** as a Filter with Pipeline Commands
- Demonstration

**Perform Week 11 Tutorial**

- Investigation 2
- Review Questions (**Parts C and D**)

## AWK UTILITY

**Definition / Purpose**

***Awk*** *is mostly used for* ***pattern scanning*** *and* ***processing****. It searches one or more files to see if they contain lines that* ***matches*** *with the specified patterns and then performs the associated* ***actions****.*

Reference: https://www.geeksforgeeks.org/awk-command-unixlinux-examples/

The awk command is useful for reading **database files** to produce **reports**.

# AWK UTILITY

## Usage

```
awk [-F] 'selection _criteria {action}' file-name
```

**How it Works:**

- The **awk** command reads all lines in the input file and will be exposed to the **expression** (contained within **quotes**) for processing.
- The expression (contained in quotes) represents **selection criteria**, and action to **execute** contained within braces `{ }`
- if selection criteria is **matched**, then **action** (between braces) is **executed**.
- The **–F** option can be used to specify the default field delimiter (separator) character eg. `awk –F";"` (would indicate a semi-colon delimited input file)

# AWK UTILITY

## Usage

```
awk [-F] 'selection _criteria {action}' file-name
```

**Selection Criteria:**

- You can use a regular expression, enclosed within slashes, as a pattern.

  For example: `/pattern/`

- The `~` operator tests whether a field or variable matches a regular expression.

  For example: `$1 ~ /^[0-9]/`

- The `!~` operator tests for no match.

  For example: `$2 !~ /line/`

## AWK UTILITY

### Usage

```
awk [-F] 'selection _criteria {action}' file-name
```

**Selection Criteria:**

- You can perform both numeric and string comparisons using relational operators ( `>` , `>=` , `<` , `<=` , `==` , `!=` ).
- You can combine any of the patterns using the Boolean operators `||` (OR) and `&&` (AND).
- You can use **built-in variables** (like `NR` or "record number" representing line number) with comparison operators.

  For example: `NR >=1 && NR <= 5`

# AWK UTILITY

## Usage

```
awk [-F] 'selection _criteria {action}' file-name
```

**Action (execution):**

- Action to be executed is contained within braces `{ }`
- The **`print`** command can be used to display text (fields).
- You can use parameters like `$1`, `$2` to represent **first field**, **second field**, etc. The parameter `$0` represents all fields within a **record** (line).
- You can use **built-in variables** (like `NR` or "record number" representing line number

  eg. `{print NR,$0}` (will print record number, then entire record)

## AWK UTILITY

### Example 1

```
cat data.txt

Saul Murray professor

David Ward retired

Fernades Mark professor
```

```
awk '{print}' data.txt

Saul Murray professor

David Ward retired

Fernades Mark professor
```

$\longleftarrow$ If no pattern is specified, awk selects **all lines** in the input

## AWK UTILITY

### Example 2

```
cat data.txt
Saul Murray professor
David Ward retired
Fernades Mark professor
```

```
awk '/^[F-Z]/ {print}' data.txt
Saul Murray professor
Fernades Mark professor
```

You can use a regular expression, enclosed within slashes, as a pattern.

In this case, the pattern is matched at the BEGINNING of each line (record) read from the input file.

## AWK UTILITY

### Example 3

```
cat data.txt

Saul Murray professor

David Ward retired

Fernades Mark professor
```

```
awk '/^[F-Z]/' data.txt

Saul Murray professor

Fernades Mark professor
```

If no action is specified, awk copies the selected lines to standard output

# AWK UTILITY

## Using Variables with awk Utility

You can use parameters which represent fields within records (lines) within the expression of the awk utility.

The parameter `$0` represents all of the fields contained in the record (line).

The parameters `$1`, `$2`, `$3` … `$9` represent the first, second and third to the 9th fields contained within the record. Parameters greater than nine requires the value of the parameter to be placed within braces
(for example: `${10},${11},${12},` etc.)

Unless you separate items in a print command with a **comma**,
awk **catenates** them.

# AWK UTILITY

## Example 4

```
cat data.txt
Saul Murray professor
David Ward retired
Fernades Mark professor
```

```
awk '$1 ~ /^[F-Z]/ {print}' data.txt
Saul Murray professor
Fernades Mark professor
```

The parameters **$1**, **$2**, **$3** … **$9** represent the first, second and third to the 9th fields contained within the record.

```
awk '$3 ~ /retired/ {print}' data.txt
David Ward retired
```

The ~ operator tests whether a field or variable matches a regular expression

## AWK UTILITY

### Example 5

```
cat data.txt

Saul Murray professor

David Ward retired

Fernades Mark professor


awk '$3 !~ /retired/ {print}' data.txt

Saul Murray professor

Fernades Mark professor
```

The `!~` operator tests for no match.

## AWK UTILITY

**Example 6**

```
cat customer.dat
A100 Acme-Inc. 5400
R100 Rain-Ltd. 11224
T100 Toy-Inc. 3413
awk '$3 > 10000 {print}' customer.dat
R100 Rain-Ltd. 11224
awk '$3 <= 6000 {print}' customer.dat
A100 Acme-Inc. 5400
T100 Toy-Inc. 3413
```

Using relational operators with the awk command.

## AWK UTILITY

**Example 7**

```
cat customer.dat
A100 Acme-Inc. 5400
R100 Rain-Ltd. 11224
T100 Toy-Inc. 3413
awk '$3 >= 5000 && $3 <= 10000  {print}' customer.dat
A100 Acme-Inc. 5400
awk '$3 <= 5000 || $3 >= 10000  {print}' customer.dat
R100 Rain-Ltd. 11224
T100 Toy-Inc. 3413
```

Using the `&&` and `||` conditional operators with the awk command.

# AWK UTILITY

### Example 8

```
cat customer.dat
A100 Acme-Inc. 5400
R100 Rain-Ltd. 11224
T100 Toy-Inc. 3413
awk '$3 > 10000 {print $1,$2}' customer.dat
R100 Rain-Ltd.
awk '$2 ~ /Acme-Inc./ {print $3}' customer.dat
5400
```

Using parameters to specify fields with print command to display output.

# AWK UTILITY

**Other Variables for awk Utility**

The table below show other variables that can be used with the awk command.

| | |
|---|---|
| **FILENAME** | **Name of the current input file** |
| **FS** | Input field separator (default: SPACE or TAB) |
| **NF** | Number of fields in the current record |
| **NR** | Record number of the current record |
| **OFS** | Output field separator (default: SPACE) |
| **ORS** | Output record separator (default: NEWLINE) |
| **RS** | Input record separator (default: NEWLINE) |

## AWK UTILITY

**Example**

```
cat customer.dat
A100 Acme-Inc. 5400
R100 Rain-Ltd. 11224
T100 Toy-Inc. 3413
awk '{print NR,$0}' customer.dat
1 A100 Acme-Inc. 5400
2 R100 Rain-Ltd. 11224
3 T100 Toy-Inc. 3413
awk 'NR ==2 {print}' customer.dat
R100 Rain-Ltd. 11224
awk 'NR > 1 && NR < 5{print}' customer.dat
R100 Rain-Ltd. 11224
T100 Toy-Inc. 3413
```

$\longleftarrow$ Using **NR** (record number) variable with the awk utility

## AWK UTILITY

**Using awk Utility as a Filter**

Although awk can be used as a streaming editor for text contained within a text file, awk can also be used as a filter using a pipeline command.

**Examples**

```
ls | awk ‘{print $1,$2}’
```

## AWK UTILITY

### Instructor Demonstration

Your instructor will demonstrate additional examples of using the **awk** utility.

## AWK UTILITY

**Getting Practice**

To get practice to perform **Week 11 Tutorial:**

- INVESTIGATION 2: USING THE AWK UTILITY
- LINUX PRACTICE QUESTIONS (Parts **C** and **D**)