Running Advanced Text Handling Commands

Sadequl Hussain
INFORMATION TECHNOLOGIST, DIGITAL CONTENT PRODUCER

@sadequlhussain

Recap

Last module

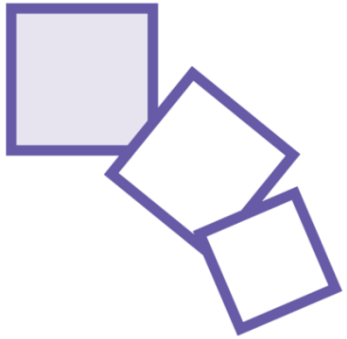
Basic text processing commands

Last two modules

Commands useful for most practical use cases

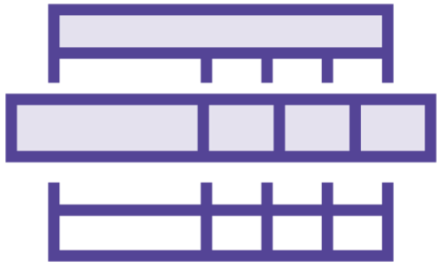
Advanced Text Processing Needs

Complex filtering or sophisticated transformations need something more powerful



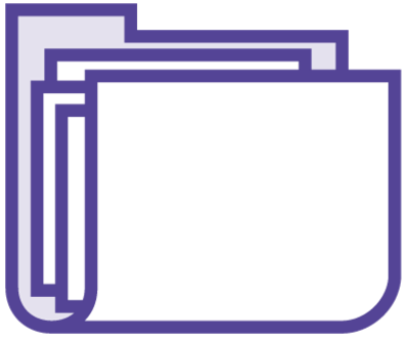
Example 1

Selectively search and replace string in text file



Example 2

Insert line at specific location in text file



Example 3

Calculate total size of files in directory

sed and awk

Part of Unix-based systems since 1970s. Also available for Linux

sed: text stream editor with powerful search and replace capability

awk: scripting language for text filtering, processing, reporting

Learning Goals

Advanced text processing techniques with sed and awk

Sample files included with exercise files

Examples

- sed in Red Hat Bash shell
- awk in Ubuntu Z shell
- Any shell is fine for learning

Let's start!

Introducing sed

sed Stream Editor

Powerful command for

- Search and replace
- Delete
- Insert
- Append
- In-place edit

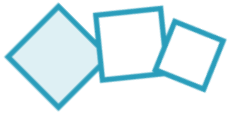
Input stream can be a file

Redirected command output

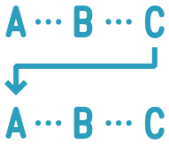
Standard input from keyboard

sed Advanced Features

Searching for text using regular expression (like grep)



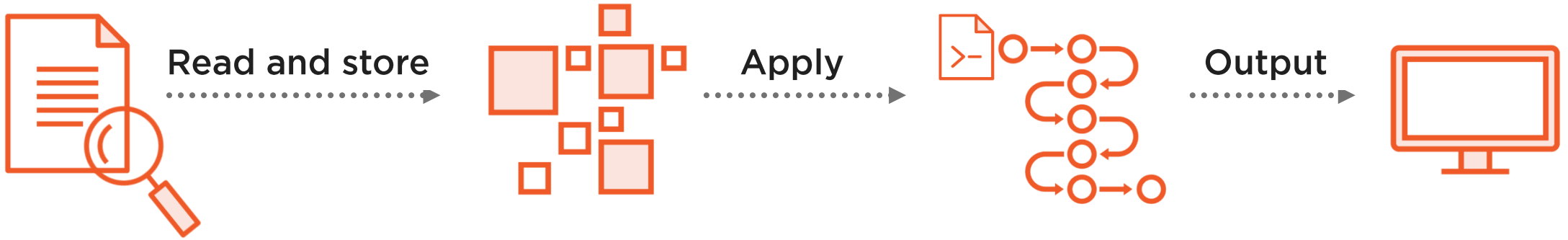
Selectively applying changes to target text



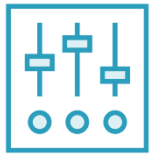
Editing file contents in single pass without opening in editor

Fast and efficient operation

sed Workflow



sed Learning Goal



Many options and commands available for sed

Allows branching and looping like programming languages

Advanced text processing with sed not covered in course

Use cases for handling common editing needs covered

```
sed -e '<inline_sed_commands>' <target_file_name>

sed -f '<sed_command_file>' <target_file_name>
```

Basic sed Syntax

Inline sed commands for course examples

```
sed -e
's/<regex_search_string>/<replacement_s
tring>/<flags>' <file_name>
```

- sed's own command is within single quotes
- "s" is for substitute
- Regular expression after the first slash ("/") is the text pattern to search for
- Text after the second slash ("/") will replace any matching text
- Flags after the third slash ("/") controls how the change will be applied
- The target file is specified in the end

Extracting Text with sed

head or tail can extract text from the start or end of a file.

How to extract text from the middle of a file?

A command's output can be piped to sed as an input stream for processing.

Introducing awk

awk is not just a Linux command.
It is a scripting language.

Not Awkward

Derived from surnames of its developers

Alfred Aho,
Peter Weinberger,
Brian Kernighan

awk
Use Cases

Not just for simple text processing

- Advanced filtering
- Data validations
- Custom transformations
- Formatted reports
- Mathematical functions

Linux shell scripts

- Conditional branching, looping, variables, arrays
- Features present in awk as well
  - Useful for specialized text handling
  - awk scripts similar to shell scripts

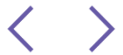
```
awk <option_1>…<option_n>… '/pattern/ {awk commands}' <target_file1>…<target_file_n>

awk <option_1>…<option_n>… -f <awk_command_file> <target_file_1>…<target_file_n>
```

Basic awk Syntax

awk can also process text from other input streams like stdin or pipes

How awk Works

An awk program is one or more pattern(s) and their associated action(s)



Patterns can be regular expressions (e.g. similar to ones used with “grep”)

Data read one line at a time from input stream and pattern matched

Associated action for the pattern performed on the matched line

Process continues until end of the input stream reached

```
{ awk commands }
```

- awk commands are specified within “blocks”, enclosed by curly brackets.

```
BEGIN { initial awk commands }
```

- An optional “starter” block is created with the keyword “BEGIN”.

```
{ main action block }
```

- The commands in the main action block are run against the target text stream.

```
/regex pattern to be matched/
```

- awk checks if there are text patterns to be matched.

  Actions performed on matching records.

  Original file remains intact.

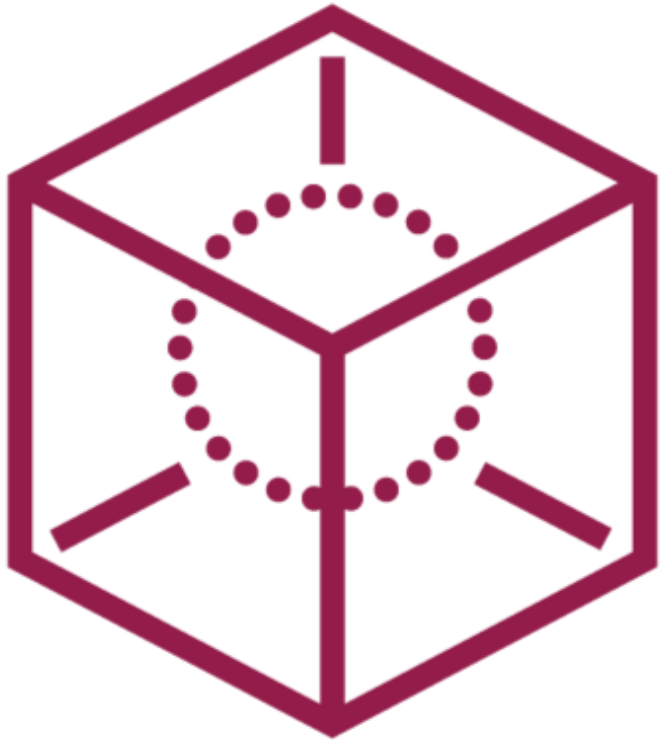
```
END { commands }
```

- Optional “END” block actions performed after main block of commands

```
BEGIN { var1=“”}{ command1; command2 }
END {print “message”}’
```

- An example of awk command blocks

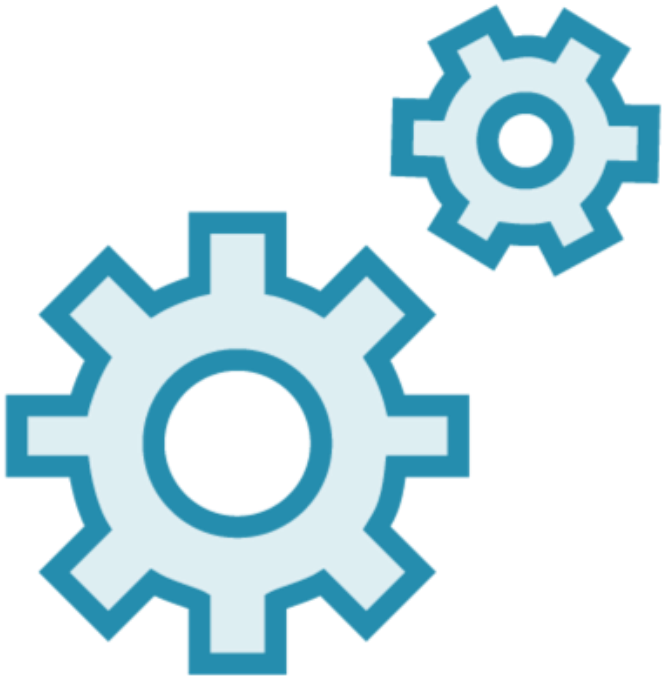
Some awk Built-in Variables



Similar to Linux environment variables

- Some can be assigned value at run time
- FS: field separator. Default value “\t”
  - awk considers each value separated by FS as a field
  - $0 = whole line, $1 = 1st field, $2 = 2nd field
- NF: Number of fields in current record
- NR: Current line number
- RS: Record separator. Default value “\n”
- FILENAME: Current file being processed

Some awk Built-in Functions



Functions for numbers, strings, I/O

- **print()**: display message on screen
- **length()**: calculate length of an input string
- **substr()**: extract portion of a string
- **gsub()**: replace part of a string with another string
- **index()**: find position of a string within another string
- **tolower()**: convert text to lowercase
- **toupper()**: convert text to uppercase

```
if (condition) command_statement;
else another_command_statement
```

- awk can run blocks of commands based on conditions. If a condition is not met, commands are not applied to target

```
‘BEGIN { FS = ","; IGNORECASE = 1 }
{ if($0 ~ "<search text>") print $2 }'
<target file>
| sed 's/search text/replace text/’
```

- Example: changing value of a specific field in a specific line of a file

```
while (condition) command_statement

do command_statement while (condition)
```

- awk can run commands repeatedly on a text stream based on conditions using “while” or “do...while” statements

Demo

awk command examples

Sample file

- currencies.csv
- Included with exercise files
- Several rows
- Two fields
  - Country name
  - Currency
- Download to /tmp/data

Summary

sed and awk

- Different from other text handling commands

sed vs. awk

- sed is simpler to use
  - Effective for searching, filtering and processing text with simple commands
- awk is a powerful scripting language
  - More versatile than sed

sed and awk Use Cases

sed is useful for pattern matching and replacements

- Example:
  - Replacing numeric values with their text equivalent (i.e. “two-hundred” instead of 200)

awk is useful for structured data (i.e. data with rows and columns)

- Examples:
  - Replacing a specific piece of data
  - Swapping column positions
  - Calculating sum of column values

sed and awk Working Together

Use Case:

- Text file column contains wrong values for some rows
- Extract rows with wrong column value to a new dataset
- Replace wrong values with correct ones in new dataset

Solution:

- Use awk to extract rows
- Use sed to replace wrong values

Summary

Other concepts learnt

- Variables and conditional branching
  - Different from basic text handling commands

Next module

- Build on the advanced concepts and take it further