Alternative OS – Lecture 8

Plan:

1. Using **grep** with regular expressions
2. **sed** – the Stream EDitor

Using grep with regular expressions

The **grep** utility is used to search Unix files for **G**eneral **R**egular **E**xpression **P**atterns (G.R.E.P.).

Syntax:

grep [options] regexp [file(s)]

Options:

- **–i** ignore case (ABC=abc)
- **–c** count occurrences and report only the number of occurrences, do not print the lines themselves
- **–v** invert the search, display only the lines that do not match the search pattern
- **–n** display line numbers, normally only lines without their numbers are displayed
- **–s** work silently, report only final status:
  - 0, if a match was found
  - 1, if no matches were found
  - 2, if an error occurred
- **–l** list only names of the files where matches were found

This command takes a *regular expression* "regexp" and a *set of file names* (may be just one file name) "file(s)" as parameters. The **grep** utility prints all lines matching the *regular expression* "regexp", it also shows which file the matching was found in.

For example:

%**grep** 'foo' bar.txt bar2.txt

If you issue this command, all lines from “bar.txt“ and “bar2.txt” containing “foo” will be displayed.

% **grep** -c 'foo' bar.txt

The **–c** option will make **grep** output how many times the line ‘foo’ was found in the file ‘bar.txt’.

% **grep** 'f[ou]' bar.txt

This way we search for all lines containing the character **f** followed by either of **o**, **u**.

Now we’ll search for all lines that **begin** with ‘**f’**:

% **grep** '^f' bar.txt

And now, for all lines that **don’t begin** with ‘**f’**:

% **grep** '^[^f]' bar.txt

This alsocould also be done by using the **-v** option:

% **grep** -v '^f’ bar.txt

Next we search for all lines that **begin** with the characters **a through e**:

% **grep** '^[a-e]' bar.txt

The next example demonstrates how to search for any instances of **b** followed by **zero or more** occurrences of **e**:

% **grep** 'be*' bar.txt

Following command will search for any instances of **b** followed by **one or more** occurrences of **e**:

% **grep** 'bee*' bar.txt

Next we show how **grep** utility can be used to search in the output of another utility rather than in a file.

Here we report on any lines output by the ***who*** program that begin with the letter **s**.

```
% who | grep '^s'

student ttyp1 Oct 12 11:41
```

sed – the Stream Editor

When a shell script is being executed the shell works in the so-called *non-interactive* mode. That means the user is limited in interacting with the shell. He receives the control over the course of actions only when the script allows it, and only as much as the script permits.

Editing task, however, is commonly attributed as an interactive one. Is it possible to edit a file from a shell script? It turns out the answer is positive!

The non-interactive, *stream editor* called **sed** edits the input stream line by line, making the specified changes and sends the result to standard output.

Syntax:

sed *[options] edit_commands [file]*

where *edit_commands* **syntax is:**

[address1[,address2]][function][arguments]

The address entries are optional. They can be separated from the commands with tabs or spaces.

- If both the **address1** and the **address2** are given, then the **address1** specifies the first line and the **address2** specifies the

last line to apply the commands.

- If only the **address1** is given then the commands are applied to the lines matching the address.
- If no address is specified then the commands are applied to the whole text.

The address can be given in two forms:

- **Line-number** is a decimal number. The last line number can be specified by the **$** character.
- **Context address** is a regular expression enclosed in slashes /.

The most often command used is the *substitution command*. We shall learn it next.

Syntax:

s/*regular_expression_pattern*/*replacement_string*/*flag*

This function tells the **sed** to find all strings matching the *regular_expression_pattern* and replace them with the *replacement_string*. This function must be quoted with single quotes (‘) if additional options are specified or functions.

Example:

```
#! /bin/sh
# converts arg[1].* files to arg[2].*
if [ $# -lt 2 ]
then
 echo 'Usage: rename_base base1 base2';
 exit 1;
fi
echo "Processing basenames  $1 and $2"
for f in $1.*
do
 newfile=`echo $f | sed s/$1/$2/`
 echo "renaming $f to $newfile"
 mv $f $newfile
done
```

Conclusions

So far we learnt usage of two different kinds of patterns: *wildcards* and *regular expressions*.

- We use *wildcards* to match characters and sets of characters in **filenames**,
- We use *regular expressions* to match characters and sets of characters in **files**.

The *wildcards* help us

- Manipulate many files at once,
- Select only those files we would *like* to be affected by the commands we submit,
- Perform other great deeds.

The *regular expressions* help us

- Finding lines even in *very* big texts,
- Finding files containing something since long lost in the piles of files,
- Counting lines that match a pattern,
- Also do great Unix deeds .

Next we learnt how to work with **sed** – the Stream EDitor. This tool let us perform editing tasks from within a script! The most useful function of **sed** is substitution. It is also noteworthy that **sed** supports *regular expressions*. Thus making it a very powerful text processing utility.

Acknowledgement

The **grep** examples using the *num.list* file were taken with minor corrections from the *"Introduction to Unix"* by Frank G. Fiamingo, Linda DeBula and Linda Condron.