IT 341: Introduction to System Administration
Using *sed*

- Maintaining Configuration Files
- The **sed** Stream Editor
- Instructions to **sed**
  - Replacing a String with **sed**
  - Adding a Line with **sed**
  - Deleting a Line with **sed**
- Inserting Text at the Beginning of a File
- Making More Than One Change with **sed**
- Using Character Ranges with **sed**
- Changing File Text Inline with **sed**

Maintaining Configuration Files

- Unix system administrators spend much of their time working with text files
- Text files need to be configured for most major Unix services and these files they need to be maintained
- In any working network things are always changing and the text files that configure services must change too
- Most of these changes are straight-forward add a line delete line or change a name

Maintaining Configuration Files

- Changing the name of something often involves making changes in more than one configuration file
- If you fail to make the needed changes in all affected files some important services will stop working
- System administrators need a way to make changes quickly and accurately in many files at the same time
- The *sed* stream editor helps to automate this task

The *sed* Stream Editor

- *sed* allows you to automate the process of editing a file
- With *sed* you can write a line or two of code that will make multiple changes to a file
- *sed* takes input from one source makes changes to the text from that source and sends the altered text to ***standard output***
- *sed* is called a stream editor because it can be used in a pipeline to transform a stream of data

The *sed* Stream Editor

- *sed* takes input from its second argument or standard input and sends the transformed text to standard output
- *sed* can be used to modify a file by redirecting standard input to come from that file and redirecting standard output to go to a new file
- The general form of a *sed* command is

```
sed  'SED_INSTRUCTIONS'  INPUT_FILE  >  OUTPUT_FILE
```

Instructions to *sed*

- The first argument to *sed* is a *sed* command telling it how to modify the file
- These commands take many forms and should be contained in single quotes
- You will find a good introduction to the *sed* commands at http://www.grymoire.com/Unix/Sed.html
- A glance at this page will show that *sed* commands can be complex

Instructions to *sed*

- But while *sed* has a lot of power and can do many things most people only use a few simple commands
- *sed* is most frequently used to:
  - *Substitute* one string for another
  - *Add* a line
  - *Delete* a line

Replacing a String with *sed*

- The most common use of *sed* is to replace one string with another
- The format for the *sed* command to do this is

```
s/OLD_STRING/NEW_STRING/
```

- The **s** tells *sed* you want it to substitute
- The **/** is used to set off two patterns
- The first pattern is what needs to be replaced and the second pattern is what it will be replaced with

Replacing a String with *sed*

- If we have a text file day.txt containing

```
day
I am looking for daylight
night and day
day after day after day
```

- We could replace instances of "day" with "night" with the following

```
sed  's/day/night/'  day.txt
```

Replacing a String with *sed*

- Running this, we get

```
$ sed  's/day/night/'  day.txt
night
I am looking for nightlight
night and night
night after day after day
```

- The output from *sed* goes to standard output unless you redirect it
- Notice the last line. Only the first instance of "day" was replaced with "night"

Replacing a String with *sed*

- Normally the **s** command only replaces the **first** string it finds on a line
- If we want *sed* to replace every string on the line we must follow the last pattern with the **g** option

```
$ sed  s/day/night/g  day.txt
night
I am looking for nightlight
night and night
night after night after night
```

- The **g** stands for global

Delimiting Patterns

- A **delimiter** is a sequence of one or more characters used to mark the boundary between different parts of a string
- The delimiter most often used to mark off patterns in *sed* is the slash, /
- But this can cause problems in certain situations
- In Unix and Linux, / is used to separates the names of directories in a path

/courses/it341/f13/ghoffmn

Delimiting Patterns

- How could you use *sed* to replace "**`f13/ghoffmn`**" with "**`s14/abird`**" in the path above using **`/`** as a delimiter?
- But you don't have to use **`/`**
- The **`s`** command uses the **very next character** as the delimiter
- So if course_directory.txt contained the path above I could change the text with the following *sed* instruction

```
's|f13/ghoffmn|s14/abird|'
```

Delimiting Patterns

- You have to put the instructions in single quotes, '' because | is the pipe character and has special meaning to the shell

```
$ cat course_directory.txt
/courses/it341/f13/ghoffmn

$ sed 's|f13/ghoffmn|s14/abird|' course_directory.txt
/courses/it341/s14/abird
```

Adding a Line with sed

- *sed* can be used to add a line of text to a file
- The format for this *sed* instruction is

```
/STRING/a\
LINE_OF_TEXT
```

- The `\` turns off the special meaning of the newline which is to signal to the shell that you are done with the command
- This command will add the line of text after **every** line that contains STRING

Adding a Line with sed

- When you write a *sed* instruction to add a line you must include all of the instructions inside single quotes
- To add a line after the second line of day.txt

```
day
I am looking for daylight
night and day
day after day after day
```

Adding a Line with sed

- …we could use *sed* like this:

```
$ sed ' /looking/a \
> and looking and looking \
> ' day.txt
day
I am looking for daylight
and looking and looking
all the live long day
night and day
day after day after day
```

Adding a Line with sed

- Notice that after I hit the Return or Enter to move down to the second line the shell prints >
- This is the **secondary prompt** that you get when you continue a command onto another line
- Notice that I had to put the name of the file day.txt outside the quote
- The second \ *turns off* the special meaning of newline at the end of the added text – allowing me to add a newline character at the end of my text

Adding a Line with sed

- If I did not do this, the new line would be inserted on the same line as the text that follows

```
$ sed ' /looking/a \
> and looking and looking' day.txt
day
I am looking for daylight
and looking and lookingall the live long day
night and day
day after day after day
```

Adding a Line with sed

- sed will add text after **every** line that matches a pattern

```
$ sed ' /day/a \
> a new line of text \
> ' day.txt
day
a new line of text
I am looking for daylight
a new line of text
all the live long day
a new line of text
night and day
a new line of text
day after day after day
a new line of text
```

Adding a Line with sed

- You can add more than one line in this way as long as you put a \ at the end of each line

```
$ sed '/looking/a \
> first line after looking\
> second line after looking\
> third line after looking\
> ' day.txt
day
I am looking for daylight
first line after looking
second line after looking
third line after looking
night and day
day after day after day
```

Deleting a Line with *sed*

- You can also use *sed* to delete a line
- The delete instruction has the following format

```
/STRING/d
```

- where STRING is some text that can **only** be found on the line to be deleted
- To delete the last line of day.txt…we could use:

```
day
I am looking for daylight
night and day
day after day after day
```

```
$ sed /after/d day.txt
day
I am looking for daylight
night and day
```

Inserting Text at the Beginning of a File

- The **a** command add text **after** a line which contains some string
- But what if you wanted to insert the text **before** a line which contains a string?
- Then you would have to use the **i** command
- If we have the file fox.txt that contains

```
The quick brown fox
jumped over
the lazy dogs.
```

Inserting Text at the Beginning of a File

- …and wanted to add a line at the **beginning** of the file we could do it like this:

```
sed '/quick/ i\
> Here is something you have seen many times:' fox.txt
Here is something you have seen many times:
The quick brown fox
jumped over
the lazy dogs.
```

Making More Than One Change with *sed*

- Each *sed* command changes one thing
- If you wanted to make several changes with a single command line entry you can use pipes
- If we have the file colors.txt

```
red blue green
light red light blue light green
dark green dark red
```

Making More Than One Change with *sed*

- …we can capitalize all the colors with the following:

```
sed 's/red/Red/' colors.txt | sed 's/green/Green/' | sed 's/blue/Blue/`
Red Blue Green
light Red light Blue light Green
dark Green dark Red
```

Using Character Ranges with *sed*

- *sed* let's you use ranges of characters to match strings
- Let's say we have the following in the file letters_digits.txt

```
abcd 1234 5678
```

- If we wanted to replace every instance of the digits 1 through 4 with the single character x we could write

```
sed 's/[1-4]/x/g' letters_digits.txt
abcd xxxx 5678
```

Using Character Ranges with *sed*

- Notice the **g** I had to add to the end of the substitution command
- If I left this out, I would get

```
sed 's/[1-4]/x/' letters_digits.txt
abcd 234 5678
```

Changing File Text *in place* with `sed`

- The default behavior of **`sed`** is to print the changes made to standard output
- However, you may also use the **`-i`** option to tell **`sed`** to make the change ***in place***, inside the file itself
- Consider the file **`roses.txt`**:

```
$ cat roses.txt
Roses are red,
And violets are blue.
Sugar is sweet,
and so are you.
```

Changing File Text *in place* with `sed`

- Now, execute this command:

```
sed -i 's/Roses/Poinsettias/' roses.txt
```

- You will not see the usual output, but if you look at the file text again, it will be ***changed***:

```
$ cat roses.txt
Poinsettias are red,
And violets are blue.
Sugar is sweet,
and so are you.
```

Changing File Text *in place* with `sed`

- In addition, you may want to save a backup copy of the original, which you can do by specifying a **suffix** when using the **`-i`** option:

```
sed -i.bk '/violets/a\(In that case...\
    ...should you not call them blues?)' roses.txt
```

- First, you will now notice that there is a ***new*** file:

```
$ ls
roses.txt       roses.txt.bk
```

Changing File Text *in place* with `sed`

- The ***original*** file will reflect the changes made:

```
$ cat roses.txt
Poinsettias are red,
And violets are blue.
(In that case...
        ...should you not call them blues?)
Sugar is sweet,
and so are you.
```

- The ***backup*** file, however, will show the *original* file text

```
$ cat roses.txt.bk
Poinsettias are red,
And violets are blue.
Sugar is sweet,
and so are you.
```