Advanced UNIX Tools

Student Workbook

itcourseware
THE SOURCE FOR IT PROFESSIONALS

Advanced UNIX Tools

Jeff Howell

Published by ITCourseware, LLC, 7245 South Havana Street, Suite 100, Centennial, Colorado 80112

Contributing Authors: Jim McNally, Rob Roselius.

Special thanks to: Many UNIX instructors whose ideas and careful review have contributed to the quality of this workbook and the many students who have offered comments, suggestions, criticisms, and insights.

Contents

Chapter 1 - Course Introduction

Notes

Course Objectives

- Take advantage of advanced features of the `vi` editor to increase productivity when editing files and programs.
- Use regular expressions to find patterns in text files. This allows you to locate specific files or text, display desired lines in files, or edit text by adding to it, deleting from it, or modifying it.
- Edit files using the streaming editor, `sed`.
- Write `awk` programs to create reports and edit text files.
- Design and write sophisticated shell scripts to automate common tasks.

Notes

Course Overview

- **Audience**: This course is designed for experienced UNIX users. It is intended to further increase the skills and productivity of programmers, system administrators, help desk staff, documenters, testers, and other "technical" folks who use UNIX frequently to do their job.

- **Prerequisites**: An introductory course in UNIX and some additional hands-on experience.

- **Classroom Environment:**

  - UNIX system with one terminal per student.

Notes

Suggested References and Reading

Dougherty, Dale and Arnold Robbins. 1997. ***sed & awk***. Second Edition. O'Reilly & Associates, Sebastapol, CA. ISBN 1565922255.

Friedl, Jeffrey E. F. 1997. ***Mastering Regular Expressions***. O'Reilly & Associates, Sebastapol, CA. ISBN 1565922573.

Lamb, Linda and Arnold Robbins. 1998. ***Learning the vi Editor***. Sixth Edition. O'Reilly & Associates, Sebastapol, CA. ISBN 1565924266.

Peek, Jerry D., Tim O'Reilly, and Mike Loukides. 1997. ***UNIX Power Tools***. Second Edition. O'Reilly & Associates, Sebastapol, CA. ISBN 1565922603.

Robbins, Arnold. 1999. ***vi Editor Pocket Reference***. O'Reilly & Associates, Sebastapol, CA. ISBN 1565924975.

Robbins, Arnold. 1999. ***UNIX in a Nutshell, System V Edition***. Third Edition. O'Reilly & Associates, Sebastapol, CA. ISBN 1565924274.

http://www.edu.physics.uch.gr/~danalis/manuals/vi/faq_frames.html

http://www.math.fu-berlin.de/~guckes/vi/

http://www.thomer.com/thomer/vi/vi.html

http://infofarm.affrc.go.jp/~kadosawa/vi.htm

Notes

CHAPTER 2 - EX AND VI OPTIONS

OBJECTIVES

- Explain the difference between **ex** and **vi**.
- Use various editor options to customize your editing environment.
- Set options during a **vi** session.
- Automatically set options at **vi** startup.

EX AND VI - TWO EDITORS IN ONE

- **ex** and **vi** are the same program file on disk.
- **ex** - extended line-oriented text editor.
- **vi** - visual (screen-oriented) text editor.
- Invocation (**ex** or **vi**) determines the initial "personality" of the editor.
- **ex** mode provides a colon (**:**) prompt; **vi** mode uses a cursor to indicate the position within the file.
- In **vi** mode, <Shift>q switches to **ex.**
- In **ex** mode, **vi** switches to visual mode.

EX AND VI - OPTIONS

- Options are variables that affect the operation of **vi** and **ex** during editing sessions.

- Options are either boolean (on or off) or they get set to a value.

  - Boolean options:

```
number
autoindent
ignorecase
```

  - Options that must be set to a value:

```
wrapmargin=8
shiftwidth=3
```

- Options are displayed with the **set** command:

  - `:set` ← Show non-default option settings.
  - `:set all` ← Show all option settings.

- Options can be set with their abbreviations:

  - **ai** = **autoindent**
  - **nu** = **number**
  - **wm** = **wrapmargin**

- Multiple options can be set at one time:

```
:set nu ai wm=8
```

- Options are set with the **set** command or in a *.exrc* file.

There are also several **vi** options that change what **vi** shows you on the screen.

showmode is an option that enables a mode prompt on the lower right of the screen. It will be blank for command mode, but will show **INSERT** mode, **APPEND** mode, etc. when you are in those modes.

list is often useful when looking at files where you need to see the position of some of the special characters. **:set list** will show all tabs as **^I** and all newline characters (end-of-line in UNIX text files) as **$**.

HOW TO SET OPTIONS WITHIN VI

- **:set nu** Displays line numbers.
- **:set ai ic** Sets **autoindent** so that each line in insert mode is indented to align with the previous line. Also in the same command, sets **ignorecase** to ignore case in searches, i.e., letters are mapped to lowercase for comparisons.
- **:set wm=5** Set **wrapmargin** to **5** spaces from right of screen.
- **:set sw=3** Set **shiftwidth** to **3** for use by **>, >>, <, <<,** and **^d** for "outdenting."

❋ **Boolean options a**re unset with the **no** prefix:

➢ **:set nonu noai noic** - Unset **nu**, **ai**, and **ic**.

Some of these options are confusing as to what they do and how they work. For instance, **wrapmargin** is somewhat confusing as to where it is setting the margin and exactly how it works when you get to the margin.

wrapmargin sets a right-hand margin for your **vi** session as a function of the right edge of the screen. If you are working in an 80 column screen and you set **wrapmargin** (also abbreviated as **w**) to **10**, when the cursor reaches the 70th column (80 - 10 = 70), **vi** will automatically append a newline character to the line at that point. This has two effects; it creates the margin at that point on the screen, and it also breaks the line into two lines. The default (**wm=0**) will not move in the margin and will not break up the line.

This can get more complicated on an X-Terminal, as the user can resize the window that **vi** is working in at any time. The act of enlarging (or shrinking) the window changes the area in which **vi** can now display, but **vi** never knows that the window has changed size unless the user uses the **resize** command to calculate the new number of rows and columns that are in the window.

Following a **resize**, **vi** will now treat a **wrapmargin** command against the new right-side column of the screen, not against the default of 80 columns. This means that in order to use **wrapmargin**, you must not only know what value it is set to, but the column size of the screen at a given time.

HOW TO SET OPTIONS IN .EXRC

- Options can be set in a *.exrc* (**.**ex runtimecommands) file or in an environment variable called **EXINIT**.
- Do not put a colon (:) in front of **set** commands in *.exrc* or **EXINIT**.

Example *.exrc* file:

```
set ai wm=5
set nu
```

Example **EXINIT** line in *.profile*:

```
EXINIT="set ai wm=5 nu"; export EXINIT
```


The algorithm used by **vi** and **ex** to set options at runtime is a little bit complicated, but not difficult to understand. It is designed to give you different editing environments for different purposes, such as programming versus writing memos. Support for this design depends upon creating *.exrc* files in different directories, and then invoking the editor from within the various directories.

Here is the algorithm:

If the environment variable **EXINIT** exists and is not null, then attempt to use its value as a set string and ignore *.exrc*. (**EXINIT** is typically set and exported in *$HOME/.profile*. If it does not contain a legal set string, then **vi** will display an error on startup, but *$HOME/.exrc* is still ignored!)

Otherwise if the environment variable **EXINIT** does not exist (or exists, but is empty), then read *$HOME/.exrc*. (**vi** is happy if *$HOME/.exrc* does not exist.)

After one of the two choices above is made, we continue with the algorithm.

If you are not in *$HOME* when you invoke **vi**, and a *.exrc* exists in the current directory, and the boolean option **exrc** is set, then read *.exrc* in current directory. Note that the boolean option **exrc** must have been set in **EXINIT** or *$HOME/.exrc*.

Here is how you set the boolean option **exrc**:

```
set exrc
```

Pretty simple, eh?

LABS

❶ Edit *Memos/orals*. Set **autoindent**. Use the **open** command (**o**) in *vi* to insert lines below an indented line. How do you "un-indent" a line in insert mode (i.e., how do you insert a line below an indented line so that the new inserted line begins in a column to the left of the existing line)?

❷ When writing C code in **vi**, it can be helpful to set the **shiftwidth** option equal to the number of columns that you are using for indenting your structured code. Edit *Progs/hitkey.c*, **set sw=3**, then add else-if blocks to handle cases for **c**, **d**, and **q** (quit). Use the <Ctrl>d technique to outdent while in insert mode.

❸ Experiment with the **wrapmargin** option. This option is provided to facilitate creating memos and other text files so that you can type rapidly without being concerned whether you are coming to the end of a line. When combined with the **spell** and **fmt** macros that we will study later, you can use **vi** as a primitive word processor.

❹ Set **ignorecase** and search for some patterns in *Memos/orals*. When would this option be useful?

❺ Make a directory and change to it. Create a *.exrc* file in this new directory and set an option that is NOT set in *$HOME/.exrc*. Invoke **vi**. Is the **local** option set?

Now set the option **exrc** in *$HOME/.exrc*. Invoke **vi** again in the new directory. Is the **local** option set now?

This feature allows you to set up different editing environments, typically programming and writing memos.

❻ Set up a *.exrc* in *$HOME* with your favorite options. Verify that they are being set when you enter **vi**. Now put a set string in the variable **EXINIT**, export **EXINIT**, and invoke **vi** again. What happened to the *.exrc* settings?

❼ In **vi**, look at the difference between **:set** and **:set all**.

Chapter 9 - Introduction to sed

Notes

Chapter Objectives

- Understand what the streaming editor *sed* is used for.
- Be able to invoke *sed* from the command line or with a *sed* script.
- Be able to use *sed* to modify text files.

Notes

There are many ways to manipulate text in UNIX. You can do it manually with an editor, although this does not lend itself to automating tasks very well. *sed* is one of the very powerful utilities available to do text manipulation, as we shall see. Others include simple utilities like *tr*,*cut*,*sort*, etc. and more complex ones like `awk` (which we will investigate later on in this class) and *perl*.

About sed

- *sed*, the streaming editor, transforms text.
- *sed* is non-interactive.
- *sed* applies one or more editing commands to the input stream.
- A set of *sed* edit commands is called a script.
- *sed* understands REs.

Examples:

`sed "s/top/TOP/g" file`	Change each (`g`) occurrence of top to TOP on every line.
`sed "s/top/TOP/" file`	Change first occurrence of top to TOP on every line.
`sort file | sed "/^$/d"`	Delete all empty lines in input stream.

Notes

sed, like almost all UNIX utilities, makes modifications to a stream of data that it processes. It has the capability to open and read a file that you give it as an argument. Do not forget that *sed* does not change the original data file! In order to save the edited data, you must use I/O redirection to capture the data stream to a file, as in

```
sed "s/top/TOP/" datafile > newfile
```

Do not attempt to redirect to the datafile in one step! If you try

```
sed "s/top/TOP/" datafile > datafile
```

the shell will do the redirection *before sed* is started, and *sed* will be working on an empty file!

Why Use sed?

- *sed* can make multiple edits to many files with one command (think of 20 changes to each of 100 files).
- Large files are processed quickly.
- Editing actions can be automated with *sed* and shell programs.
- Very powerful tools are built around *sed*.

Notes

You are allowed to place one comment line in a *sed* script. It must start with the # character, as in

```
# This is a comment line
s/Joseph/Joe/g
s/William/Bill/g
```

sed will recognize only one formal line of comments. If you need more than one line, you can expand to many lines by using a backslash as the last character of each line up to the last line of your comments, as in

```
# This is a comment line\
this is the second line of comments, and if you wish,\
this is the third line
s/Joseph/Joe/g
s/William/Bill/g
```

This limitation of one comment line at the top of the *sed* script is present in all System V UNIX variants. Some Berkeley variants may allow multiple comment lines to be placed anywhere in the script. This is nice (and probably a good idea in complex scripts) but beware of losing portability if the script needs to run on both Berkeley and System V.

Invoking sed

- *sed* can be invoked in one of three ways:
  - `sed` *`script`* `[` *`file`* `... ]`
  - `sed -e` *`script`* `-e` *`script`* `... [` *`file`* `... ]`
  - `sed -f` *`scriptfile`* `[` *`file`* `... ]`
- The first specifies a single script (one command).
- The second specifies a multi-part script.
- The third directs *sed* to get its script from a file.
- In each case, if no file is specified, `sed` reads from standard in.

```
sed -e "s/Joseph/Joe/g" -e "s/William/Bill/g" list
```

If the file *informal.sed* contains:

```
s/Joseph/Joe/g
s/William/Bill/g
```

then we can say:

```
sed -f informal.sed list
```

Notes

Whenever you need to use *sed* to manipulate data, it is important to realize just how *sed* does its job. This can be covered in three basic steps:

- All of the commands in a script are applied in order to each line of the data before moving on to the next line of data.

- Commands are applied to all lines of data unless line addressing is used to restrict this.

- The source data file is left intact; the output of *sed* is sent to standard out, and must be redirected by the shell if it is to be captured.

How sed Works

- *sed* "reads in" a line, performs any specified edits on the line, then writes the (possibly) modified line to standard out.

  Consider:

```
sed -e "s/chevy/chevrolet/g" -e "/ford/d" <cars >cars.new
```

  Explanation (in pseudo-code):

```
while not EOF (on cars)
   read in the next line of text (from cars)
   if the line contains the pattern chevy
      change it to chevrolet
   if the line contains the pattern ford
      do not write the line to standard out
   else
      write the line
end while
```

Notes

The “pattern space” referred to on the facing page is a part of the internal mechanism of *sed*, a detailed discussion of which is beyond the scope of this course. You can think of the “pattern space” as an internal buffer where *sed* holds the line while it is performing edits on it; something like a work-space.

There are other “spaces” used for other purposes in the *sed* program. For more information on these and other internal mechanisms of *sed*, refer to a more advanced text, some of which are listed in the reference section at the beginning of this workbook.

Once again . . .

- *sed*:

    1. Reads a line into the pattern space.
    2. Checks if the line should be edited.
    3. If it should, then edits the line.
    4. Writes the line to standard out, whether it was edited or not, unless an edit function (such as delete) prevents the line from being written.

- *sed* performs the above "cycle" until end of input.
- *sed* scripts can actually change *sed*'s default actions somewhat (as we will see) but the underlying cycle still applies.

Notes