Text Editing in UNIX

A short introduction to vi, pico, and gedit

About UNIX editors

- There are two types of text editors in UNIX: those that run in terminal windows, called ***text mode editors***, and those that are ***graphical***, with menus and mouse pointers. The latter require a windowing system, usually X Windows, to run.
- If you are remotely logged into UNIX, say through SSH, then you should use a text mode editor. It is possible to use a graphical editor, but it will be much slower to use. I will explain more about that later.

Text mode editors

- The three text mode editors of choice in UNIX are **`vi`**, **`emacs`**, and **`pico`** (really **`nano`**, to be explained later.)
- **`vi`** is the original editor; it is very fast, easy to use, and available on virtually every UNIX system. The **`vi`** commands are the same as those of the **`sed`** filter as well as several other common UNIX tools.
- **`emacs`** is a very powerful editor, but it takes more effort to learn how to use it.
- **`pico`** is the easiest editor to learn, and the least powerful. **`pico`** was part of the Pine email client; **`nano`** is a clone of **`pico`**.

What these slides contain

- These slides concentrate on **vi** because it is very fast and always available. Although the set of commands is very cryptic, by learning a small subset of the commands, you can edit text very quickly.
- What follows is an outline of the basic concepts that define **vi**. It is not a tutorial; it does not contain specific instructions on the commands in **vi**. It is intended to supplement a tutorial, which sometimes fails to describe the big picture. I have written a tutorial, which you can download here.

About `vi`

- **`vi`** (***pronounced vee ai***) is the original, full-screen editor for UNIX systems. It has served the UNIX community for more than thirty years, having been written by Bill Joy when he was a graduate student at UC Berkeley in 1976.
- Before **`vi`** was written, the UNIX text editor was a line-oriented editor called **`ex`**. **`vi`** extended the functionality of **`ex`** to give it full-screen capabilities. **`vi`** got its name because to get from **`ex`** to the full-screen visual editor, you had to type "**`vi`**".
- **`vi`** is required to be in all versions of UNIX by the Single UNIX Standard.

`vi` Versus `vim`

- Through the early 1990's most UNIX users used `vi` or `emacs`. (`emacs` was written in 1976 by Richard Stallman.)
- But an "improved" version of `vi` called `vim` (`vi` ***IMproved***) was written and released in 1991 by Bram Moolenaar. `vim` is upward-compatible from `vi`, so that everything in `vi` works in `vim`.
- `vim` offers more features than `vi`, but it is also much harder to learn than `vi` (unless all you are trying to do is learn the part of `vim` like `vi`!)

Running **`vi`**

- On the system you are using, when you type the **`vi`** command in **`bash`**, **`vim`** runs. If you want the original **`vi`** program, you are out of luck. Many people consider **`vim`** to be better than **`vi`**, so it has become standard to install **`vim`** instead of **`vi`**.

Caveat

- In these slides, unless I write otherwise, what I am describing is **`vi`**, not **`vim`**. That is, if I say something can be done in **`vi`**, it means it can be done in **`vi`** and hence in **`vim`** also.
- If I say something cannot be done, it cannot be done in **`vi`** but it is possible it can be done in **`vim`**.

Notation

- **<CR>** denotes the character obtained by pressing the ***Enter*** key on the keyboard.
- **<SP>** will denote the ***space*** character, obtained by pressing the ***space bar***. Sometimes it will be written as an underline "_" instead of **<SP>**.
- **<ESC>** will represent the character obtained by pressing the ***escape*** key.
- A phrase enclosed in angle brackets **< >** represents the character(s) described by the phrase. For example, **<*positive integer*>** will mean any positive integer, and **<*char*>** will mean any single character.

Modality and finite state machines

- `vi` has three ***modes***, or ***states.*** At any given time it is in exactly one of these modes. With each key that you press, it either stays in its current mode or changes to a different one.
- Things that have a finite number of states and that change their state depending on what events take place are called ***finite state machines***, or ***finite automata***. Vending machines are finite automata. So are most video games.
- As an example, a car is always in a specific gear. It can be neutral, first, second, third, fourth, perhaps fifth, and in reverse. Some also have a "parked" gear. The driver or the car itself (in automatic transmissions) changes gears.

Starting *vi*

- To edit a file named **myfile**, at the command prompt, type

  vi myfile

- When **vi** starts, the file's contents are copied into a ***buffer***. A buffer is just a temporary storage area.
- If you just type **vi** without a filename after it, **vi** starts up with a new, empty buffer.
- When **vi** starts it will be in ***Command Mode***.
- **vi** is a 3-state finite automaton, with states

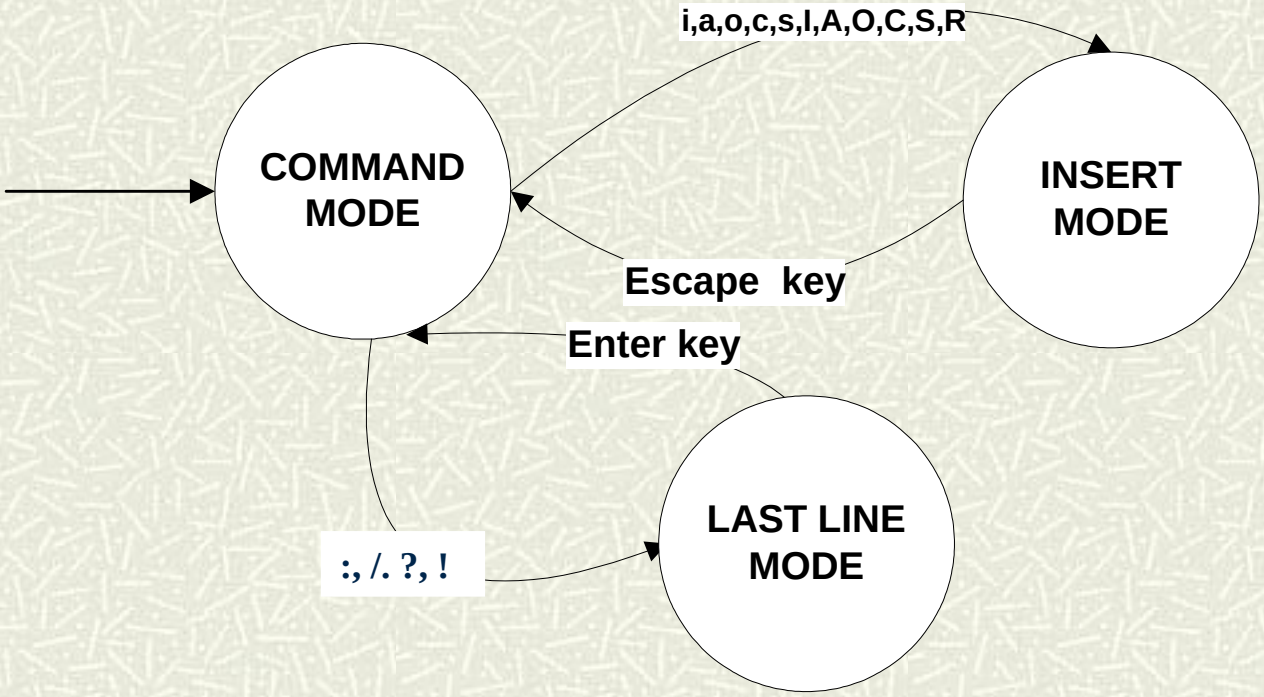
  Command Mode

  Insert Mode

  Last-line Mode

A Conceptual model of **vi**

- This is a picture of the way that **vi** works. The arrows from one state to another are labeled by the keys that cause its state to change.



Some general rules

- **`vi`** is ***CASE-SENSITIVE***: everything you type must be in the correct case.
- **`vi`** makes a copy of the file in a temporary location to use as its work buffer. Sometimes this copy is readable by other people who know how to find it. This security flaw has been corrected in modern UNIX systems.
- The cursor in **`vi`** is always ***on*** a character, ***not between*** characters.
- The ***start of a line*** is the leftmost ***NON-WHITE-SPACE*** character in that line.
- The ***end of a line*** is the last character before the ***<CR>***, even if it is a white space character.

Getting help on **`vi`** and **`vim`**

- There are no on-screen help facilities in **`vi`**.
- To get help on-screen while using **`vim`**, type **`:help`** and follow the instructions on the screen.
- You can and should read the **`vimtutor`**, which is an application that displays a document describing how to use **`vim`**.
- There are many **`vi/vim`** tutorials on the Web. You can always do a search for them and use them while you are working.

Insert Mode

- In ***Insert Mode***, everything you type is inserted into the text at the text insertion point, until you type **<ESC>**. For this reason, there is very little you need to learn about Insert Mode.
- Control characters can be inserted by typing **Ctrl-V** first and then the control character. E.g., to put a **Ctrl-M** into a file, you type **Ctrl-V** followed by **Ctrl-M**.
- It is easy to tell when **vi** is in Insert Mode because it will display a line at the bottom of the screen with the word

```
-- INSERT --
```

Last-Line Mode

- ***Last-Line Mode*** has an odd name, but it is because when **`vi`** is in that mode, there is a prompt on the last line of your screen.
- Last-Line Mode is entered only from ***Command Mode***, and only by pressing either '**`:`**', '**`/`**' or '**`?`**'.
- '**`/`**' and '**`?`**' tell **`vi`** to search in the file for a pattern that you provide after it followed by a **`<CR>`** character. The '**`/`**' searches forward and the '**`?`**', backward.

Last-Line Mode

- The '`:`' tells **`vi`** to expect a command. There are several types of commands, such as those to
  - read a file into the buffer after the current line,
  - write the current contents to a file,
  - do multiple-line operations such as replacements, deletions, copying to buffers,
  - set line markers or view marked lines,
  - view special characters in lines, and
  - run a shell command in a subshell.

Command Mode

- The only way to know that **`vi`** is in ***Command Mode*** is to realize that it is not in the other modes. If you do not see the word "**`INSERT`**" and you do not see the prompt for Last-Line mode (**`:`**,**`/`**,**`?`**), then it is in Command Mode.
- In Command Mode you can navigate through the file, make changes within single lines, and copy parts of lines or paste into them. Of course Command Mode is the ***base camp***, so it is from here that you can enter Insert Mode or Last-Line Mode.
- Command Mode also lets you create and run macros and abbreviations, redraw the screen, and do much more.

The most important commands

- ***NAVIGATION***: Commands to move the cursor left, down, up, right are **`h`**,**`j`**,**`k`**, and **`l`**; the ***arrow keys*** usually work as well. Other keys move the cursor to various places, such as **`^ $ ( ) { } w e b`**
- ***DELETION***: Commands to delete words or characters include **`x`** and **`d`**. Whole lines are deleted with **`dd`**.
- ***SUBSTITUTION***: Substitutions are made by typing **`s/`*pattern*`/`*replacement*`/`** within a line.
- ***SEARCHING***: Searches within a single line use **`f`** or **`t`**. To search in the whole file, enter Last-Line Mode with **`/`** or **`?`**.

Insertion commands

- The most basic and important commands are those that let you enter Insert Mode in various ways:
  - **`i`** insert to the left of insertion point
  - **`a`** insert to the right of insertion point
  - **`o`** insert a line below current line
  - **`O`** insert a line above current line
  - **`I`** insert at beginning of line
  - **`A`** insert at end of line

  There are others. These are the simplest to use.

The Temporary Buffer

- **`vi`** has a ***temporary buffer*** that it uses like a clipboard. It lets you copy text into the buffer using **`y`** (yank) or **`yy`** (to copy multiple lines) and paste text (**`p`** or **`pp`**) from the buffer into specific points in the file.
- It also makes a copy of text deleted with the delete command (**`d`** or **`D`**) in the buffer, so that it can be pasted also.
- But unlike the clipboard, the buffer text cannot be retrieved unless you do so ***immediately*** after storing it there.

Command repetition

- Most commands can be preceded by a positive integer to affect their behavior in some way. For example:
- The command **`w`** advances to the next word in the file from the cursor, and **`5w`** advances to the **`5th`** word from the cursor.
- The **`a`** command followed by text and then **`<ESC>`** adds the text to the right of the insertion point, so **`8a+====<ESC>`** adds the string **`+====`** 8 times:

  `+====+====+====+====+====+====+====+====`

  all at once.
- In general, a number ***`n`*** preceding a command repeats that command ***`n`*** times.

Getting started

- You should read one of the tutorials I have made available: either my own tutorial, or the **`vimtutor`**, or one of the many websites that do the same thing.

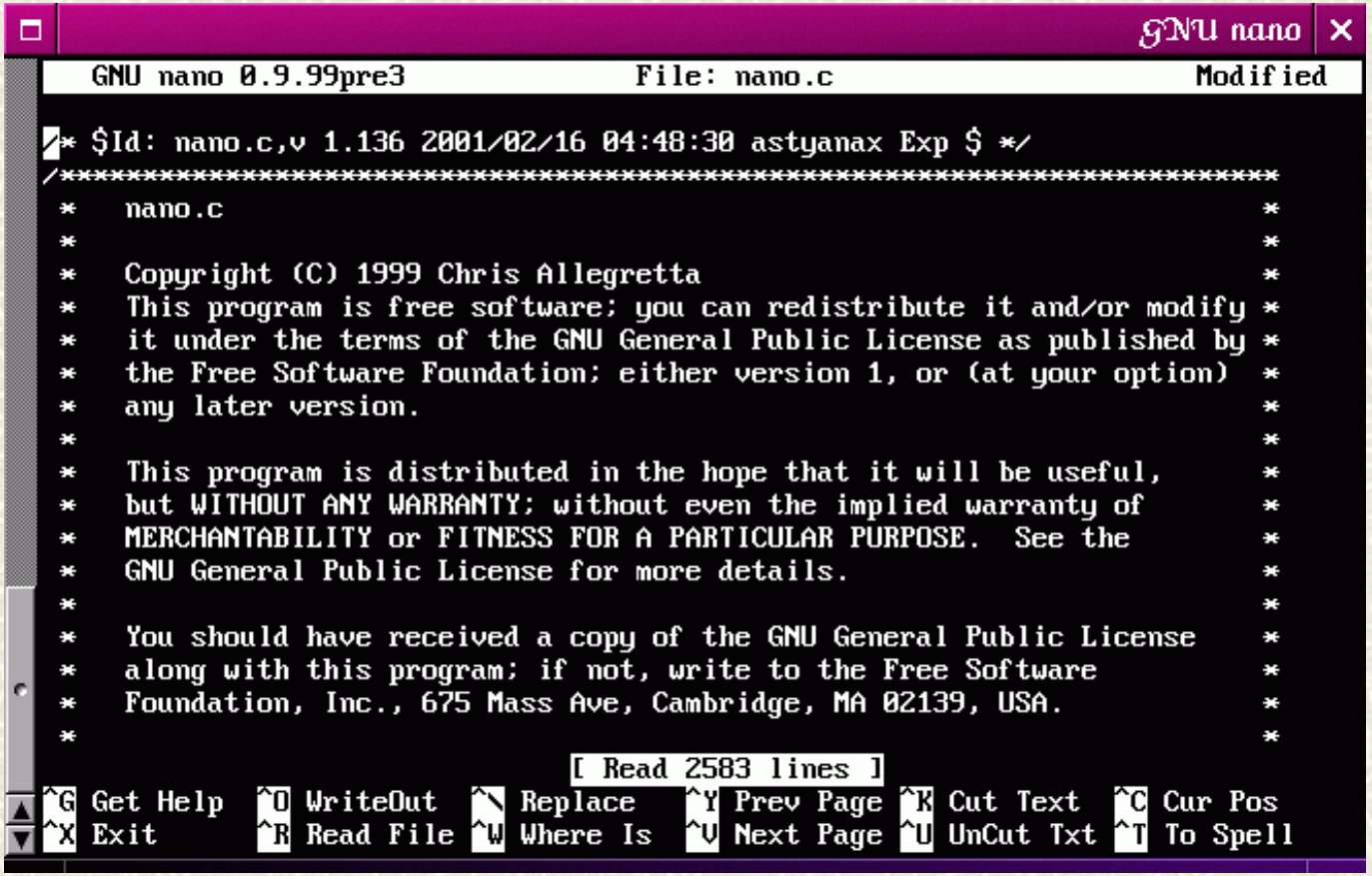
About pico

- Pico was the message-composition editor in ***Pine***, a UNIX mail reader developed by the University of Washington. (Pico = ***Pi***ne ***Co***mposer.)
- Pine was not a free product, and was distributed as part of Pine. GNU developed a free, even smaller and faster editor that essentially incorporates all of the features of **`pico`** and adds a few more. Like all GNU products, it was named carefully, and is called **`nano`**. (**`pico`** is now free under the Apache License.)
- In the next few slides I briefly describe the key features of **`nano`**, which are essentially the same as those of **`pico`**.

Using nano and pico

- To edit a file using **`nano`**, type **`nano <filename>`**.
- In **`nano`**, there is only one mode, which is equivalent to **`vi`**'s Insert Mode. Everything typed is entered into the document.
- The ***arrow keys*** move the cursor around the screen.
- All commands are of the form *Ctrl-<char>*. In **`nano`** (and **`pico`**) at the bottom of the screen there are reminders of the common commands. They use the notation **`^C`** for *Ctrl-C*. ***I will use that notation here.***
- It is pretty self-explanatory; to move down by one screenful, use **`^V`**; to move up, use **`^Y`**. To delete a line, **`^K`**.

nano screenshot



More about nano

- **nano** has a large number of command line options that control how it behaves. You should read the man page to learn about the more useful ones.
- At any time, pressing **^G** gives you online help.
- There are ways to enter characters that are not on the keyboard (*Esc-Esc-ascii-code* does it, e.g. **Esc-Esc-007** enters the bell character.)
- To save a file, i.e., to write the buffer to disk, use **^O**.
- To exit, **^X**.
- There are other features, but these are the basic ones.

Graphical editors

- If you are working on the UNIX console, you can choose from a large assortment of editors that have graphical user interfaces. Among these you may find *nedit* and *gedit*.
- gedit is like Windows *Notepad*; when it starts up you have the text on the screen and can use point and click just as in Notepad or Wordpad.
- It has many more features that make it far more powerful than tools like Notepad, such as built-in parsers. If you are editing a Perl program for example, it parses it for you and can format it as well. I use it to edit html because it displays all of the tags in different colors and makes it easy to edit.

About **gedit**

- **gedit** also has a spellchecker and other tools, but what I find most useful are the snippets, which are like ***plug-ins*** that extend its capability.
- For example, for **html** documents, there are snippets that can add templates for any tag, as well as form fields and more.
- One can create new snippets for it as well. Overall, gedit is good choice of editor when logged onto the UNIX host so that you are working in a desktop environment (GUI), unless you have learned **emacs**, which is even more powerful.
- So why not use it all of the time?

When not to use `gedit`

- **`gedit`** requires X Windows to run. X Windows is a collection of software programs that allow applications to create windows, track the mouse, and treat the screen as a bitmapped display in general. You cannot run **`gedit`** in a terminal window!
- There is a way to run it across a network using SSH, but it will be terribly slow if you are at home, many miles from the UNIX server, and it will require that you install an X Windows server on your Windows or Mac local computer.

What is an X Windows server?

- This is a topic beyond this course. Essentially you have to put software on your local computer that does the job that would have been done had you been sitting in front of the UNIX machine in the lab. Roughly speaking, it recreates the X Window environment on your non-UNIX operating system.
- My recommendation is to not bother; all those bits of a bit-mapped display have to travel back and forth a long way just to edit a plain text file, and it is not a worthwhile use of the bandwidth of the network. Use **`vi`**, **`nano`**, or **`emacs`** instead.