Development Tools 2: Text Editors

1. Intro
2. **Text Editors**
   - **vi/vim**
   - **emacs**
   - **kwrite/kate**
   - **gedit**
   - **other FOSS text editors**
   - **indentation tools**
3. IDE's
4. Compiling
5. Debugging and Tracing
6. Advanced Development

Text Editors

Programming language **source code** is just text, so any **text editor** can be used to create a source code file (this applies to all other programming languages as well).

Syntax-aware editors understand the syntax of a programming language so provide useful functions, such as:

- automatic *indentation* of code
- *syntax highlighting* (using different colors for different parts of programs)
- *macros* to simplify the insertion of different program constructs

Some editors operate in a *command line* or "*terminal window*" environment while others operate in a GUI environment.

A few are able to operate in either CLI/GUI environment, or have different versions for both environments.

Text Editors (contd.)

Command line ("terminal window") editors:

- most are **screen oriented**, though a few **line editors** are still available
- movement within a document is typically accomplished *without a mouse*, via the *arrow keys* and/or other special keystrokes
- typically make heavy use of **keyboard shorcuts**
- shortcuts may use the **control**, **alt**, **meta** keys
- able to be used *remotely* (e.g., via **SSH**)

Text Editors (contd.)

GUI editors:

- **screen oriented** of course
- movement mainly accomplished using *mouse*
- most have fairly limited set of built-in functionality
- some functionality may be accessed via **keyboard shorcuts** but most accessible only via **menus**
- can be used remotely only if able to **redirect display** to local machine (running X11)

Text Editors (contd.)

Two of the most popular and powerful text editors for Linux/UNIX are **Emacs** and **vi**.

There are several versions of each available, both can be used in *either a command-line or GUI mode*, they are syntax aware, and include many built-in commands.

Of course, from the long running **UNIX editor wars** we know: *vi blows and Emacs rules!!*

Seriously, though, these two editors are far more powerful and customizable than any text editor a student is likely to have encountered under Windows.

While they have *steep learning curves*, once one learns enough of their functionality, all other editors (and even IDEs) will seem (and in fact be) pathetic excuses for text editors.

Text Editors (contd.)

Of the two, basic vi functionality can be picked up more quickly.

Also, because vi is much smaller it is virtually always available in rescue environments, so Linux sysadmins must know it.

When editing is to be done in a command line (terminal window) environment, most people use either vi or Emacs.

Probably the next most commonly used editor is **nano** (a clone of **Pico**).

The most popular GUI-only text editors are those for KDE and GNOME: **KWrite** (KDE) and **gedit** (GNOME).

These two editors are very similar in look/feel to many Windows text editors, so should be easy for students to learn how to use.

Text Editors (contd.)

While syntax aware, they offer only a fraction of the functionality that is available in Emacs or vi.

You are probably used to such editors, but the constant moving of one hand from keyboard to mouse and back severely reduces the speed at which one can work.

The extensive functionality that vi and Emacs make available via keyboard shortcuts can result in much higher productivity.

Of course this requires the commitment to become familiar with this functionality, whereas the "GUI editors" require little effort.

vi

Linux distros generally include **vim** (vi iMproved) as their vi.

A key aspect of vi is that it is a **mode-based** editor:
keystrokes have different meaning depending on the current mode.

vi has three modes:

- **command**: keystrokes are interpreted as commands
- **insert** (**text**): everything typed is inserted into file
- **command line** (**ex**): command entry at bottom of screen, limited commands to save, exit, etc.

vi (contd.)

Keystrokes to change mode:

- `<escape>` changes from insert to command mode
- `:` (colon) changes from command to command line mode
- several keys change from command to insert mode:
  - `i` (insert before cursor)
  - `a` (insert after cursor)
  - `I` (insert at beggining of line)
  - `A` (insert at end of line)
  - `o` (open new next line)
  - `O` (open new prior line)

vi (contd.)

Here are some example vi keyboard shortcuts:

- `:q` – exit
- `:w` – write/save buffer
- `^` – move to beggining of line (text)
- `$` – move to last char in line
- `w` – move forward one word
- `b` – move backward one word
- `d` – delete highlighted text
- `dd` – delete line
- `j` – go down a line
- n – prefix most commands to be repeated n times ("`5j`")
- n`G` – goto line n
- `:help` – get help
- `:set number` – turn on line numbers
- `q`$char$ – start recording macro (`q` to end)
- `u` – undo

vi (contd.)

vim has a C plugin (c.vim) that provides functionality specifically for C development.

If this plugin is not included in your distro's version of vim, you can download it from **vim.org**.

Other plugins are also availabe to provide various IDE-like features for vim: Cscope, Ctags, code_complete, etc.

Some functionality may also be added via "scripts" in the vim startup file: `.vimrc`

vi (contd.)

There is a great deal of information about configuring and using vi/vim on the web.

Key sites related to vi/vim:

- `www.vim.org`
- `vim.wikia.com/wiki/Vim_Tips_Wiki`

Emacs

The two main versions of Emacs are **GNU Emacs** and **XEmacs**.

Both are available with most Linux distros, but neither is generally installed by default.

GNU Emacs will run in GUI mode if it detects X11 running, else it will use command-line mode.

Emacs includes *hundreds* of built-in **commands**.

Most commands are written in a version of Lisp called **Emacs Lisp** or just **Elisp**.

Customized functionality can be added to Emacs by writing new Elisp functions, making Emacs **extensible** (by users).

Emacs (contd.)

While some functionality can be accessed via the GUI *menus*, most must be accessed by invoking commands via *keyboard shortcuts* or by entering the command name.

Characteristics of Emacs shortcuts:

- **keystrokes** can include **modifier keys** like **Control**, **Meta**, and **Shift**
- a shortcut can involve *multiple keystrokes*
- provides a very large set of possible **key bindings** for commands
- key bindings can be changed by users
- not all commands will have shortcuts

Emacs (contd.)

For example, Emacs binds the command for finding/opening a file to the shortcut "`C-x C-f`" (said as "control x control f").

This means to type the "x key" while holding down the "Ctrl key" and then then type the "f key" while still holding down the "Ctrl key."

While you will often see "control x" written as "`^x`" it is written as "`C-x`" in Emacs to allow other modifiers to be denoted.

The other two modifiers you will see used by default in Emacs are **Meta** and **Shift**.

Meta is denoted with an "`M`", so "`M-x`" means hold down the "Meta key" and type the "x key."

Emacs (contd.)

Shift is denoted with an "`S`", so "`C-S-f`" means hold down both the "Ctrl key" and "Shift key" and type the "f key."

Since most PC keyboards do not have a dedicated Meta key, X11 bindings are typically set up so that the **Alt** key functions as a Meta key.

In command-line/terminal mode, the Alt key may work as Meta, or you may have to use the Escape key: "`<ESC> x`" (type "Esc" then type "x") is equivalent to "`M-x`".

`M-x` is a special shortcut: it brings up a prompt at the bottom of the screen, at which you can enter any *Emacs command*.

When prompted, commands can be *auto completed* by typing either `<Space>` or `<Tab>`.

Emacs (contd.)

Here are some basic example Emacs keyboard shortcuts:

- `C-x C-c` – exit Emacs
- `C-x C-s` – save current buffer into file
- `C-a` – move to beginning of current line
- `C-e` – move to endof current line
- `C-d` – delete character to right of cursor
- `M-d` – delete word to right of cursor
- `C-k` – delete line (point to end of line)
- `C-x (` – start keyboard macro ("`C-x )`" to end)
- `C-w` – delete highlighted text (to buffer)
- `M-w` – copy highlighted text (to buffer)
- `C-y` – yank text from buffer (and insert)
- `M-S-%` – search and replace
- `C-u` – command prefix
- `C-S-_` – undo
- `C-g` – quit command

Emacs (contd.)

Opened files are held in **buffers**, and Emacs can have an arbitrary number of files open at once.

Multiple buffers can be shown at a time in separate **panes** within the single Emacs window.

When Emacs identifies a file type that it understands, it opens the file in a **mode** specialized for the file type (e.g., C source, Java source, Bash shell, etc.).

Modes provide capabilities such as syntax highlighting, automatic indentation, useful shortcut commands, etc.

Commands and shortcuts may be "global" or specific to a mode.

Emacs (contd.)

Emacs supports C development by integrating with GCC, GDB, shells, diff, etc.

E.g., the `compile` command opens a new pane that allows clicking on errors to move to relevant lines in the source file pane.

While Emacs is certainly the most complex editor in existence, it does include extensive online help and tutorials.

Help can be accessed by typing "`C-h`" (control h).

A list of key bindings (shortcuts) can be gotten with: "`C-h` b".

Relevant commands can be found with *apropos*: "`C-h` a".

Since Emacs is so heavily customizable, users typically have a `.emacs` startup file that customizes their own Emacs environment.

Emacs (contd.)

There is a vast amount of information about configuring and using Emacs on the web.

Key sites related to vi/vim:

- `www.gnu.org/software/emacs`
- `www.xemacs.org`
- `www.emacswiki.org/emacs`
- `www.gnu.org/software/emacs/manual/html_node/emacs/index.html` (GNU Emacs manual)
- `www.gnu.org/software/emacs/manual/html_node/elisp` (GNU Email Elisp manual)

KWrite/Kate

KDE includes two GUI text editors: **KWrite** and **Kate**.

The two editors share many features, but Kate is more advanced.

Key features:

- understand C syntax so does highlighting
- indenting support (see `Tools` → `Align`)
- does **code folding**
- kate has tabs, kwrite shows single file in each window
- can view line numbers
- kate plugins provide integration with GCC and GDB
- allow remote file editing (e.g., via SSH)

gedit

gedit is the primary GUI editor for GNOME.

Key features:

- understands C syntax so does highlighting
- no indenting support
- does not do code folding
- multiple files open in *tabs*
- can view line numbers
- no integration with GCC or GDB

Indentation Tools

Keeping your developing code properly **indented** is important.

Improperly indented code can make you think your code is correct when it is not, making debugging more difficult.

For example, you might starting with code like:

```
if (x > y)
  x = 0;
```

During debugging, you find more must be done under this condition, so you add another line, indented as you want the logic to be:

```
if (x > y)
  x = 0;
  y = 0;
```

Indentation Tools (contd.)

While this code visually looks as desired, the lack of {}'s around the `if` body means that it is actually not as intended, and not correct.

If it were properly indented (according to C's logic), it would in fact look like:

```
if (x > y)
  x = 0;
y = 0;
```

That would make it much easier to detect the incorrect logic and the need for {}'s to be added.

Indentation Tools (contd.)

Many text editors and IDE's includes tools that can properly indent C source code, but some do not, or cannot indent your code in the desired/required style.

There are two main command-line tools for indenting C source code independent of an editor or IDE:

- **indent** (GNU software)
- **Artistic Style** (**astyle**)

While both can be primarily thought of as indentation tools, they both are capable of significantly *reformatting code* to adhere to a style, even if that means *moving braces*, *breaking lines*, etc.

Furthermore, because of their flexibility, there are not overly easy to get to do what you want, if you want your code to adhere to a style that differs from one of their standard styles.

indent

GNU **indent** is a widely available in Linux distros.

It has more than *80 options*.

It has four defined *styles:*

- GNU style (the default)
- Kernighan & Ritchie style
- Linux kernel style
- the original Berkeley `indent` style

Each style can be invoked by using a single option with `indent`, implicitly setting a number of "background" options.

A style's background options can be *overridden* by explicitly specifying other options.

indent (contd.)

E.g., we can indent/format a C source file according to K&R style by doing:

```
indent -kr program.c
```

This command will cause `program.c` to be indented/formatted, with a backup copy of the the original in `program.c~`.

Once one finds the options required for a desired style, they can be committed to a *profile file* to ease future use:

- file named by environment variable `INDENT_PROFILE`
- `./.indent.pro`
- `~/.indent.pro`

See `indent`'s man page or online documentation (at `gnu.org`) for further information.

astyle

astyle is a popular open source project: `astyle.sourceforge.net`

Its documentation is online: `astyle.sourceforge.net/astyle.html`

It has a large set of *styles* that affect braces (and control constructs): default, allman, java, kr, stroustrup, whitesmith, vtk, ratliff, gnu, linux, horstmann, 1tbs, google, mozilla, pico, lisp.

Other options affect indentation, padding, line breaking, etc.

E.g., we can indent/format a C source file according to K&R style by doing:

```
astyle --style=kr program.c
```

This command will cause `program.c` to be indented/formatted, with a backup copy of the the original in `program.c.orig`.

astyle (contd.)

Once one finds the options required for a desired style, they can be committed to an *options file* to ease future use.

There are a range of methods for specifying which options file `astyle` should use.

A standard approach is to use: `~/.astylerc`