

Lecture 2: Headless environment II

CEE 690

First: Customize your environment

Every time you ssh into a remote server, the `.bashrc` file is run to setup your session

The `.bashrc` defines the “experience” that you have at the terminal

What is a .bashrc file?

- Standard Bash script located in your home directory (~/.bashrc)
- It is hidden (the dot). “ls” won’t find it but “ls -a” will
- It runs every time you open a terminal or open a new ssh session

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-pagi
# export SYSTEMD_PAGER=

# User specific aliases and functions

# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/hpc/home/nc153/miniconda3/bin/conda' 'shell.bash' '"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
else
    if [ -f "/hpc/home/nc153/miniconda3/etc/profile.d/conda.sh" ]; then
        . "/hpc/home/nc153/miniconda3/etc/profile.d/conda.sh"
    else
        export PATH="/hpc/home/nc153/miniconda3/bin:$PATH"
    fi
fi
unset __conda_setup
# <<< conda initialize <<<

~
```

What can you add to a .bashrc?

- Aliases (shortcuts)

```
alias gohome='cd ~'
```

```
(base) nc153@dcc-login-03 ~/CEE690 $ cd /
(base) nc153@dcc-login-03 / $ gohome
(base) nc153@dcc-login-03 ~ $ ls
CEE690 miniconda3 Miniconda3-latest-Linux-x8
```

- Environment variables

```
# Add a custom software folder to your system path
export PATH="$HOME/my_custom_tools/bin:$PATH"

# Set your default editor to Vim (so git uses it for commits)
export EDITOR='vim'
```

- Customize the prompt

```
# A prompt that shows [User] in color
export PS1='\[\e[31m\]\u \$ \[\e[0m\]'
```

```
[nc153 $
```

Bashrc customization

- Example 1 - The traffic light

```
[nc153 $ source .bashrc
[✓ ~ $ ls
CEE690 miniconda3 Miniconda3-latest-Linux-x86_
[✓ ~ $ cd..
-bash: cd..: command not found
✗ ~ $ ]
```

- Example 2 - The Git branch

```
[nc153@dcc-login-05 ~/CEE690 (main) $ ls
Assignments Lectures LICENSE README.md
nc153@dcc-login-05 ~/CEE690 (main) $ ]
```

- Example 3 - High visibility

```
[21:52:25] nc153@dcc-login-05:~ $ ]
```

Bashrc customization

- Example 4 - Permanent log of commands

```
# 1. Ensure the history file is appended to, not overwritten
shopt -s histappend

# 2. Define the log file location (e.g., a hidden file in your home folder)
export COMMAND_LOG="$HOME/.permanent_history_log"

# 3. The magic function that captures metadata
log_command() {
    # Only log if the command isn't empty
    local last_cmd=$(history 1 | sed 's/^ *[ ]*[0-9]*[ ]*//')
    if [ -n "$last_cmd" ]; then
        echo "[$(date '+%Y-%m-%d %H:%M:%S')] [PWD: $PWD] $last_cmd" >> "$COMMAND_LOG"
    fi
}

# 4. Execute the function every time a command finishes
export PROMPT_COMMAND="log_command; $PROMPT_COMMAND"
```

```
[21:55:27] nc153@dcc-login-05:~ $ tail -n 20 ~/.permanent_history_log
[2026-01-12 21:54:40] [PWD: /hpc/home/nc153] source .bashrc
[2026-01-12 21:54:52] [PWD: /hpc/home/nc153] vi .bashrc
[2026-01-12 21:54:54] [PWD: /hpc/home/nc153] source .bashrc
[2026-01-12 21:55:10] [PWD: /hpc/home/nc153] vi .bashrc
[2026-01-12 21:55:12] [PWD: /hpc/home/nc153] source .bashrc
[2026-01-12 21:55:13] [PWD: /hpc/home/nc153] ls
[2026-01-12 21:55:15] [PWD: /hpc/home/nc153] ls
[2026-01-12 21:55:17] [PWD: /hpc/home] cd ..
[2026-01-12 21:55:23] [PWD: /hpc/home] ls
[2026-01-12 21:55:27] [PWD: /hpc/home/nc153] cd ~
```

Sourcing .bashrc

Simply editing .bashrc files is not enough... You have to load (i.e. source) it

```
[21:58:25] nc153@dcc-login-05:~ $ source .bashrc
```

GUI vs “headless” text editors

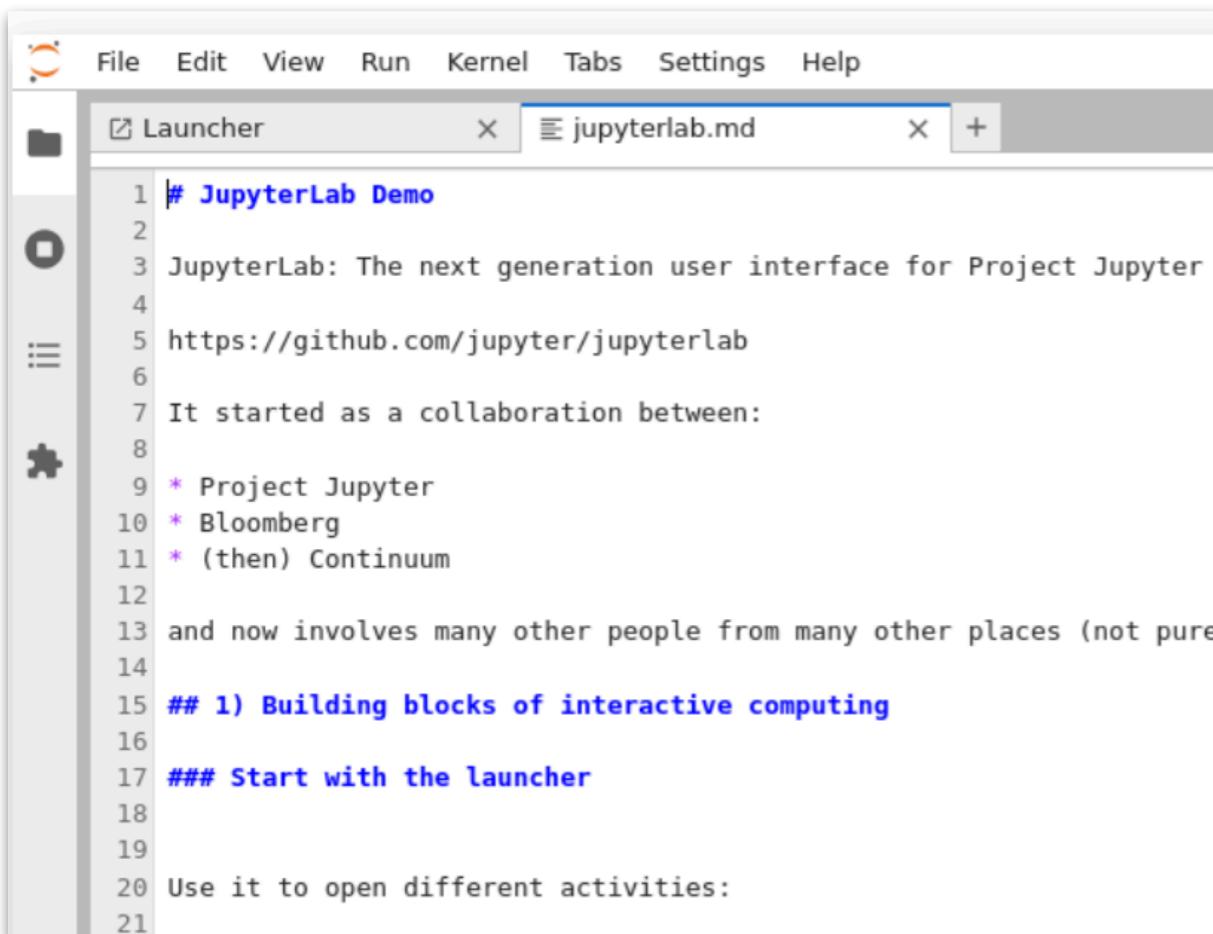
GUI

- Sublime
- VS Code
- PyCharm
- Jupyterlab
- ...

no-GUI

- Vi/Vim
- Emacs
- Nano
- ...

Or...



A screenshot of a JupyterLab interface. The top navigation bar includes File, Edit, View, Run, Kernel, Tabs, Settings, and Help. Below the bar, there are two tabs: 'Launcher' and 'jupyterlab.md'. The 'jupyterlab.md' tab is active, showing a code editor with the following content:

```
1 # JupyterLab Demo
2
3 JupyterLab: The next generation user interface for Project Jupyter
4
5 https://github.com/jupyter/jupyterlab
6
7 It started as a collaboration between:
8
9 * Project Jupyter
10 * Bloomberg
11 * (then) Continuum
12
13 and now involves many other people from many other places (not pure)
14
15 ## 1) Building blocks of interactive computing
16
17 ### Start with the launcher
18
19
20 Use it to open different activities:
21
```

Vs.

```
# User specific aliases and functions

# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/hpc/home/nc153/miniconda3/bin/conda' 'shell.bash' 'hook' 2>&1)
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
else
    if [ -f "/hpc/home/nc153/miniconda3/etc/profile.d/conda.sh" ]; then
        . "/hpc/home/nc153/miniconda3/etc/profile.d/conda.sh"
    else
        export PATH="/hpc/home/nc153/miniconda3/bin:$PATH"
    fi
fi
unset __conda_setup
# <<< conda initialize <<<

# A prompt that shows [User] in color
#export PS1='\[\e[31m\]\u \$ \[\e[0m\]'
#export PS1='${if [ $? -eq 0 ]; then echo "\[\e[033[01;32m\]\u"; else echo "\[\e[033[01;31m\]\u";}'
export PS1='[\e[1;32m]\t [\e[1;36m]\u [\e[37m]@\[\e[1;36m\]\h:\[\e[1;33m\]\w] [\e[1;34m]\n]'

alias gohome='cd ~'

# Add a custom software folder to your system path
export PATH="$HOME/my_custom_tools/bin:$PATH"
```

What are the disadvantages of GUI editors?

- Latency gap (SSH) -> GUI will freeze or hang
- Resource overhead -> GUIs are “heavy”
- Setup time -> Getting the GUIs to work takes time
- Mouse dependency
- Distraction -> Too much stuff going on
- Lack of portability

What are the disadvantages of non-GUI editors?

- Steep learning curve
- Lack of visual context
- It takes time to get a headless editor to feel modern
- No inline plots
- It requires more active thought until you are an expert

Contemporary approaches: Pass only text and commands and render locally



OnDemand provides an integrated, single access point for all of your HPC resources.

Pinned Apps A featured subset of [all available apps](#)



RStudio AppTainer
System Installed App



Jupyter Lab AppTainer
System Installed App



Jupyter Lab
System Installed App



DCC Desktop
System Installed App

Get Jupyterlab running on the compute cluster, access the terminal via there, and use the Jupyterlab text editor. All the rendering is happening on “your” side so the latency issue is non-existent.

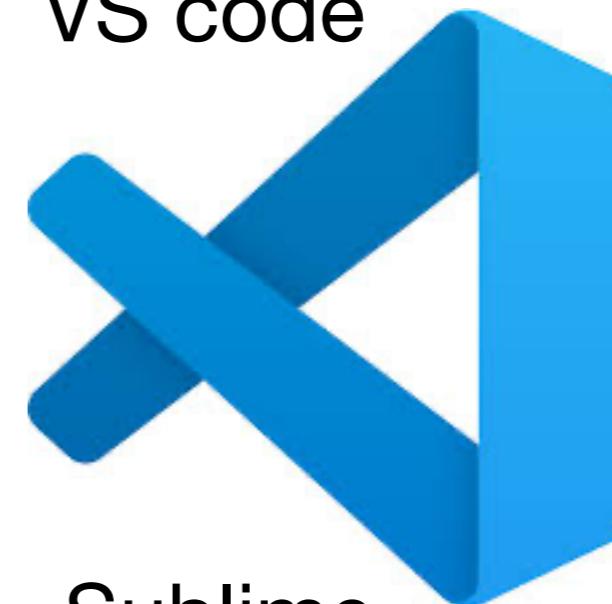
But if you come to rely on this, you will be disappointed when some remote systems don't have it

Other contemporary local editors that you can get this paradigm to work with



PyCharm

VS code



Nova



Sublime

But if you want to feel like you know what you are doing... don't use this for now

Non-GUI editors



Core Philosophy	Modal editing: 'Do one thing and do it well' (text editing).	Extensible platform: 'An OS that happens to edit text.'	Simplicity: 'What you see is what you get.'
Learning Curve	Steep. Requires memorizing a new 'language' of commands.	Very Steep. Uses complex modifier key combinations (Ctrl/Meta).	None. Commands are listed at the bottom of the screen.
Availability	Ubiquitous. Pre-installed on almost every Linux/Unix system.	Common, but often requires manual installation on servers.	Widely available on most modern Linux distributions.
Extensibility	High via VimScript or Lua (Neovim). Thousands of plugins.	Infinite. Built on Lisp; can be a mail client, web browser, etc.	Very Low. Basic syntax highlighting, but no real plugin system.
Best For	Speed, efficiency, and remote server management.	Users who want to live entirely inside one environment.	Beginners and quick, one-off configuration edits.

Thank you Gemini

History of Vim

- Early 1970s - “Ed” was the original Unix line editor
- 1976 - Bill Joy created “Vi” for visual. This revolutionized text editing
- 1991 - Bram Molenaar released “Vim” which stands for “Vi improved”
- Bram maintained Vim until he passed away in 2023.



Vim is a “right of passage”



Arrow keys - hjkl

- The ADM-3A that this was made on did not have arrow keys so Bill Joy used hjkl instead.
- You can use arrows nowadays; however, there are ergonomic and efficiency reasons why you might want to stick with hjkkl



Modal editing

Normal mode

- Navigation
- Deletion
- Copy/paste
- Cut
- ...

Insert mode

- Navigation
- Deletion
- Copy/paste
- Cut
- ...

You are always going back and forth between normal and insert mode (it's a feature not a bug)

Vim basics I

Entering/exiting

- Insert (i)
- Back to normal
(esc)
- Save and quit (:wq)
- Force quit without
saving (:q!)

Basic movement

- Right (l)
- Left (h)
- Down (j)
- Up (k)

Vim basics II

Deleting

- Character (x)
- Entire line (dd)

Undo/redo

- Undo (u)
- Redo (Ctrl + r)

The grammar of Vim

Verb + noun

- Delete + word (dw) [deletes word]
- Change + word (cw) [deletes word and puts you in insert mode]

Number + action

- Delete 3 lines (3dd)
- Jump forward 3 words (5w)

Precision navigation

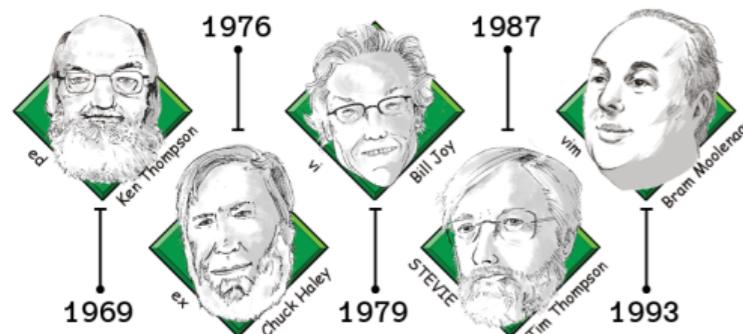
Inside a line

- 0 - Start
- \$ - End
- F{char} - Find a specific character

Inside a file

- gg - Top
- G - Bottom
- /{word} - Search

Basic Vim cheat sheet



Invocation	Legend
vim -	Read from STDIN
vim +90	Jump to line
vim -u NONE	“Debug” mode
sheet designed by Max Cantor thingsfittogther.com	
a Key press a <code>^a</code> , <code><C-a></code> , <code>Ctrl + a</code> a Hold shift to modify usage	

Registers “ < Normal Mode i s a o c > Insert Mode

d d	Cut line to default register	“	d scissors	z z	Save & Close	/	n	Next	x	Complete...	n	Autocomplete
p	Paste from default register	“	y	z Q	Close	g g		Word count	o	Jump back	n	Next
“ a p	Paste from	a	p	z w	Swap files	6			o	File	n	Previous
“ a d d	Cut line to	a	g f	Open file under cursor		*		Find word under cursor	1	Line	y	Confirm
“ E 3 y	Append 3 lines to	e	:	Command Mode		v		Visual Mode	t	Tag	r “	Paste from “
Esc or c or [Return to Normal Mode												

Options ? < Command Mode > Manual ?

<code>:set wrap lbr</code>	Notepad “mode”	Enter	Submit command	f	History]	Drill down	t	Bubble up
<code>:set ft?</code>	Check filetype	:quit	Exit Vim	:edit	Open file	:help navigation	:help :s	Search & replace	
<code>:set list et</code>	Spaces only	:write	Save file	:vsplit	Vertical	:help options	:help ^w	w indow commands	
<code>:set ru nu sc</code>	Inspect “mode”	:pwd	Current directory	:close	Close split	:help windows	:help map	Macros & keys	
<code>:set sw=4 sts=4</code>	Tab-to-space ratio	:ls	List open files	:bn	Go to file n	:help motion	:help i^w	Insert mode w ?	
<code>:set tw&</code>	Reset textwidth	:cd	Change directory	:help	Learn Vim	:help registers	:help word-motions		
<code>:set nohlsl</code>	Disable search hilighting	:set	Set options	:reg	List all registers	:help cmdline	:help pattern-searches		

Source: <https://thingsfittogther.com/>

Advanced Vim cheat sheet

This image is a detailed Vim cheat sheet, organized into several sections:

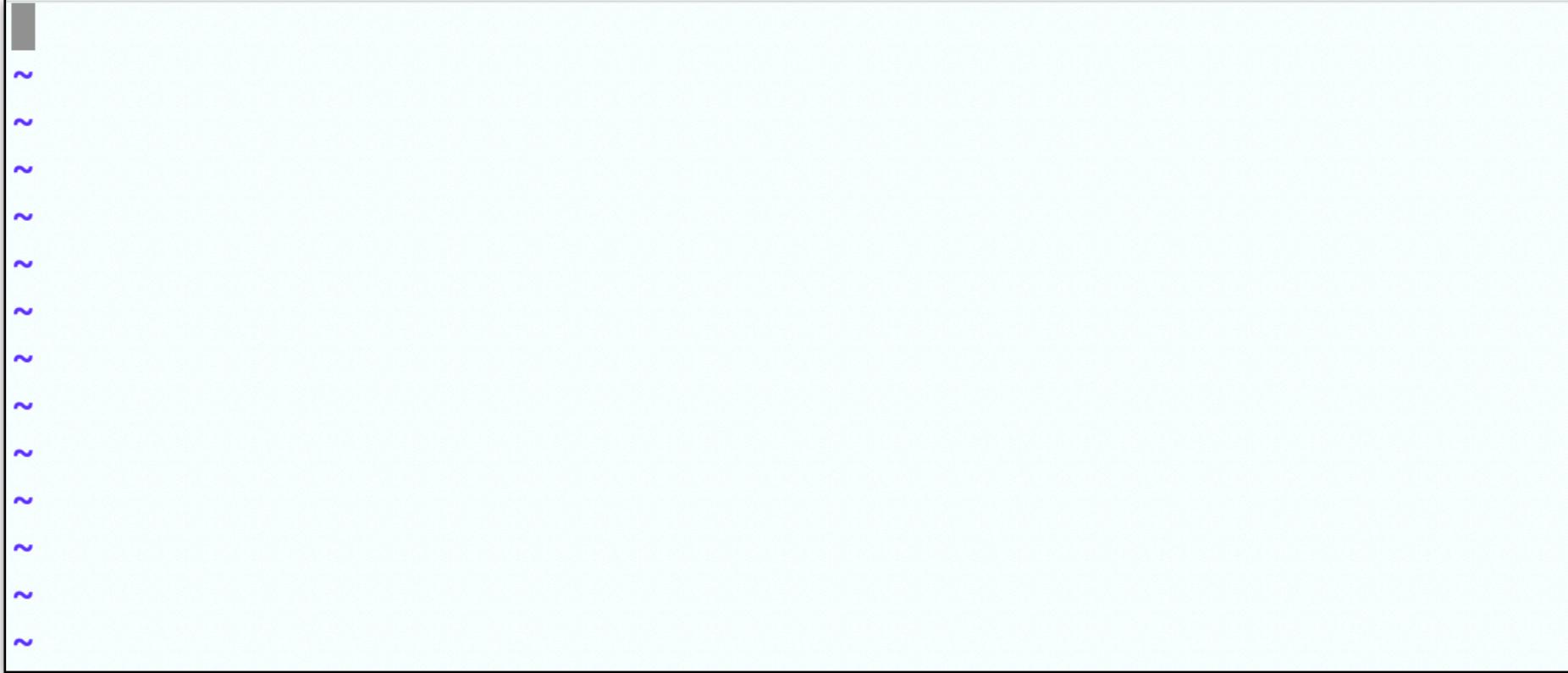
- Top Left:** A tree diagram showing the hierarchy of Vim operators: [operator][count][motion]. It includes mappings for d (delete/cut), y (yank/copy), c (change), gg (first line), ^b (up 1 page), ^u (up 10 pages), k (up 1 line), i (insert mode), and (use text-objects) iw (insert into word).
- Top Center:** A section titled "MIXING TABS AND SPACES IS RIGHT OUT." It covers :retab (Replace all tabs with spaces according to current tabstop setting), fileformat (Try changing this if your line-endings are messed up), and list (Display whitespace visibly according to listchars).
- Top Right:** A large red "vim" logo.
- Right Side:** A table of Vim key mappings categorized by function (e.g., <CR>, <Tab>, <C-n>, <M-n>, <Esc>, <BS>,).
- Middle:** A large central area containing:
 - A search section with commands like j (down 1 line), ^d (down 10 pages), ^f (down 1 page), G (last line), p (paste after cursor), P (paste before cursor), ^[(return to Normal mode), u (undo), ^r (redo), . (repeat), gf (find file under cursor in path and jump to it), dd (delete current line), yy (yank current line), x (delete character after cursor), % (jump to matching paren), r (replace char under cursor), nG (jump to line n), ^o (jump back), ^i (jump forward), zz (center screen on cursor), zt (align top of screen with cursor), zb (align bottom of screen with cursor), == (auto-indent current line), << (shift current line left by shiftwidth), and >> (shift current line right by shiftwidth).
 - A "COOL INSERT MODE STUFF" section with commands like ^w (delete word before cursor), ^u (delete line before cursor), ^rr (insert the contents of register r), ^r= (use the expression register), ^t (increase line indent by shiftwidth), ^d (decrease line indent by shiftwidth), ^x^l (line completion), and ^n (find next completion suggestion according to complete).
 - A "COMMAND-LINE MODE ONLY" section with commands like ^f (edit using Normal mode), ^r^w (insert word under cursor), ^d (completion suggestions), and various substitute and search commands.
 - A "7 words" section with links to <http://www.vimcheatsheet.com> and [1 WORD](#).
- Bottom Right:** A sidebar with tips and a table of Vim registers (labeled "REGISTERS are CLIPBOARDS").
- Bottom Left:** A large section covering:
 - SEARCHING:** Commands like N (Next), n (Forward), and b (Backward).
 - ENTERING INSERT MODE:** Commands like I (before cursor), a (after cursor), O (next line), o (substitute character), S (substitute line), s (line from cursor), and C (line to cursor).
 - ENTERING VISUAL (SELECT) MODE:** Commands like V (switch cursor to start/end), gv (re-select previous area), ^V (prepend to each Visual block line), I (jump to start of prior area), ZZ (Write current file, if modified, and quit), ZQ (Quit without checking for changes (like q!)), :write (Write current file), and :wq (Write current file and quit).
 - Syntax and Make:** Commands like :syntax (Enable and configure syntax highlighting), :make (Run a compiler and enter quickfix mode), and :! (Execute external shell command).
 - Registers:** Commands like @r (Access register r as a variable), @r@r (Access register r as a variable), and @@ (Repeat last playback).

Source: <https://thingsfittogether.com/>

Vim customization (.vimrc)

- Eliminate manual/repetitive tasks
- Tailor the editor to your language
- Create “muscle memory” shortcuts
- Portability and consistency

Vimrc: Most basic example



```
#!/bin/sh
#
# NAME: Miniconda3
# VER: py38_4.9.2
# PLAT: linux-64
# LINES: 578
# MD5: d84cff5da9dc8f4cd1a947cd13521f66

export OLD_LD_LIBRARY_PATH=$LD_LIBRARY_PATH
unset LD_LIBRARY_PATH
if ! echo "$0" | grep '\.sh$' > /dev/null; then
    printf 'Please run using "bash" or "sh", but not ." or "source"\n' >&2
    return 1
fi
```

Vimrc: Low complexity example

```
1 " --- General Settings ---
2 set nocompatible          " Use Vim defaults instead of old Vi behavior
3 syntax on                 " Enable color highlighting for code
4 set number                " Show line numbers on the left
5 set cursorline             " Highlight the current line your cursor is on
6 set showcmd                " Show the command you are typing in the bottom right
7
8 " --- Search Settings ---
9 set incsearch              " Highlight matches as you type your search
10 set hlsearch               " Keep matches highlighted after you hit Enter
11 set ignorecase             " Ignore case when searching...
12 set smartcase              " ...unless you use a capital letter
13
14 " --- Indentation & Tabs ---
```

```
1#!/bin/sh
2#
3# NAME: Miniconda3
4# VER: py38_4.9.2
5# PLAT: linux-64
6# LINES: 578
7# MD5: d84cff5da9dc8f4cd1a947cd13521f66
8
9export OLD_LD_LIBRARY_PATH=$LD_LIBRARY_PATH
10unset LD_LIBRARY_PATH
11if ! echo "$0" | grep '\.sh$' > /dev/null; then
12    printf 'Please run using "bash" or "sh", but not "." or "source"\n' >&2
13    return 1
14fi
```

Vimrc: Medium complexity example

```
1 " --- Basic Improvements ---
2 set nocompatible
3 filetype plugin indent on      " Load plugins for specific file types
4 syntax on
5 set number
6 set relativenumber           " Show line numbers relative to cursor (great for jumping)
7 set cursorline
8 set title                     " Set the terminal window title to the filename
9
10 " --- The Leader Key ---
11 " Map the leader key to 'space' (much easier to reach than backslash)
12 let mapleader = " "
13
14 " --- Search & UI ---
15 set incsearch
```

```
7 #!/bin/sh
6 #
5 # NAME: Miniconda3
4 # VER: py38_4.9.2
3 # PLAT: linux-64
2 # LINES: 578
1 # MD5: d84cff5da9dc8f4cd1a947cd13521f66
```

```
8
1 export OLD_LD_LIBRARY_PATH=$LD_LIBRARY_PATH
2 unset LD_LIBRARY_PATH
3 if ! echo "$0" | grep '\.sh$' > /dev/null; then
4     printf 'Please run using "bash" or "sh", but not "." or "source"\n' >&2
5     return 1
6 fi
```

Vimrc: High-complexity example

```
1 "# =====
2 " 1. PLUGINS
3 " =====
4 call plug#begin('~/vim/plugged')
5
6 " Aesthetics & UI
7 Plug 'joshdick/onedark.vim'          " Professional color scheme
8 Plug 'itchyny/lightline.vim'        " Lightweight status bar
9 Plug 'preservim/nerdtree'           " File explorer sidebar
10 Plug 'ryanoasis/vim-devicons'      " Icons for file types
11
12 " Navigation & Search
13 Plug 'junegunn/fzf', { 'do': { -> fzf#install() } }
14 Plug 'junegunn/fzf.vim'             " Blazing fast fuzzy finder
15 Plug 'christoomey/vim-tmux-navigator' " Seamless Vim/Tmux navigation
16
17 " Coding Intelligence
18 Plug 'neoclide/coc.nvim', {'branch': 'release'} " LSP support (IntelliSense)
19 Plug 'tpope/vim-commentary'         " Shortcut to comment code
20 Plug 'tpope/vim-fugitive'          " The best Git wrapper
21 Plug 'psf/black', { 'tag': 'stable' } " Python formatter
22
23 call plug#end()
24
25 " =====
26 " 2. CORE SETTINGS
27 " =====
28 set nocompatible
29 filetype plugin indent on
30 syntax on
31 set termguicolors                 " Enable 24-bit RGB colors
```

You will need to install plug-ins. I have never used this and didn't know it was a thing until I made this lecture.

Vimtutor

[09:15:00] nc153@dcc-login-04:~/CEE690/Miscellaneous/vimrc \$ vimtutor

```
=-----=  
= Welcome to the VIM Tutor - Version 1.7 =  
=====
```

Vim is a very powerful editor that has many commands, too many to explain in a tutor such as this. This tutor is designed to describe enough of the commands that you will be able to easily use Vim as an all-purpose editor.

The approximate time required to complete the tutor is 30 minutes, depending upon how much time is spent with experimentation.

ATTENTION:
The commands in the lessons will modify the text. Make a copy of this file to practice on (if you started "vimtutor" this is already a copy).

It is important to remember that this tutor is set up to teach by use. That means that you need to execute the commands to learn them properly. If you only read the text, you will forget the commands!

Now, make sure that your Caps-Lock key is NOT depressed and press the `j` key enough times to move the cursor so that lesson 1.1 completely fills the screen.

Lesson 1.1: MOVING THE CURSOR

** To move the cursor, press the `h,j,k,l` keys as indicated. **

^

k Hint: The `h` key is at the left and moves left.

Learn Vim?

