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" ]; the
        . "/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

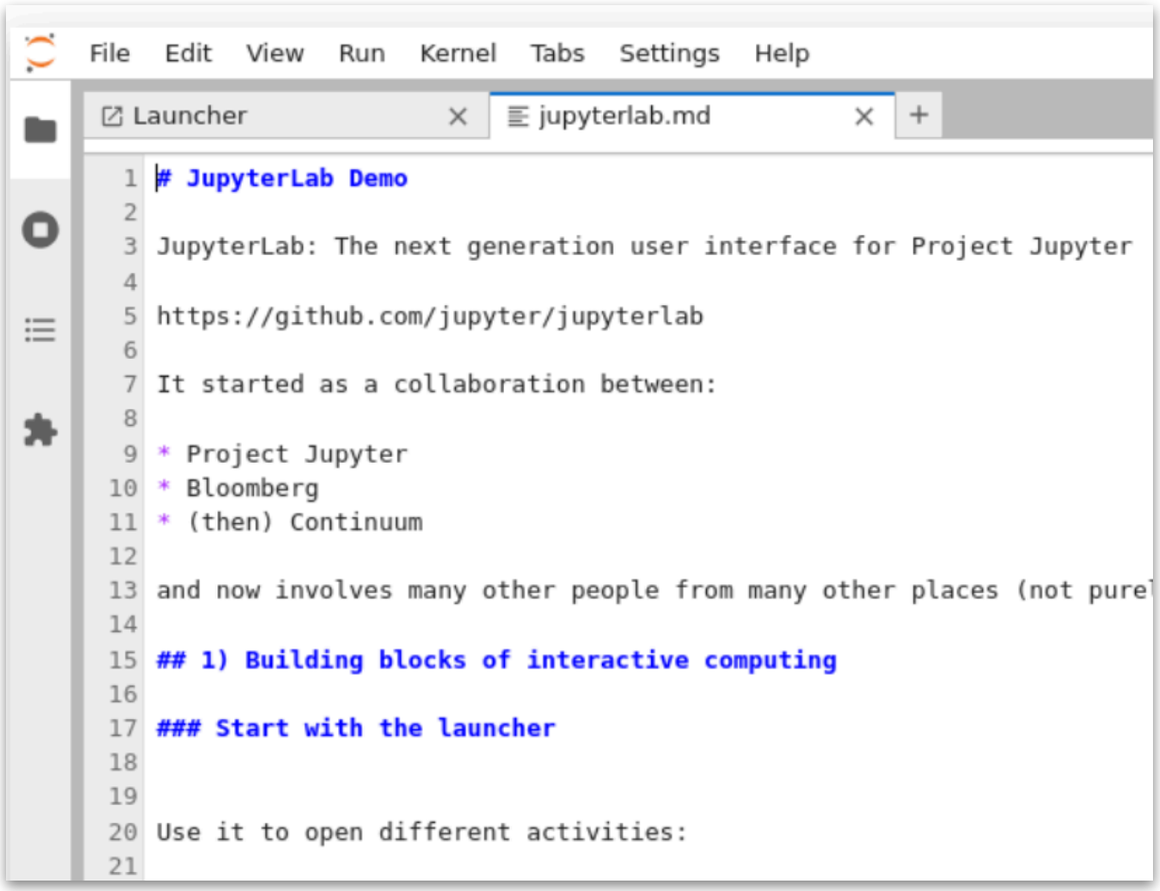
GUI

- Sublime
- VS Code
- PyCharm
- Jupyterlab
- …

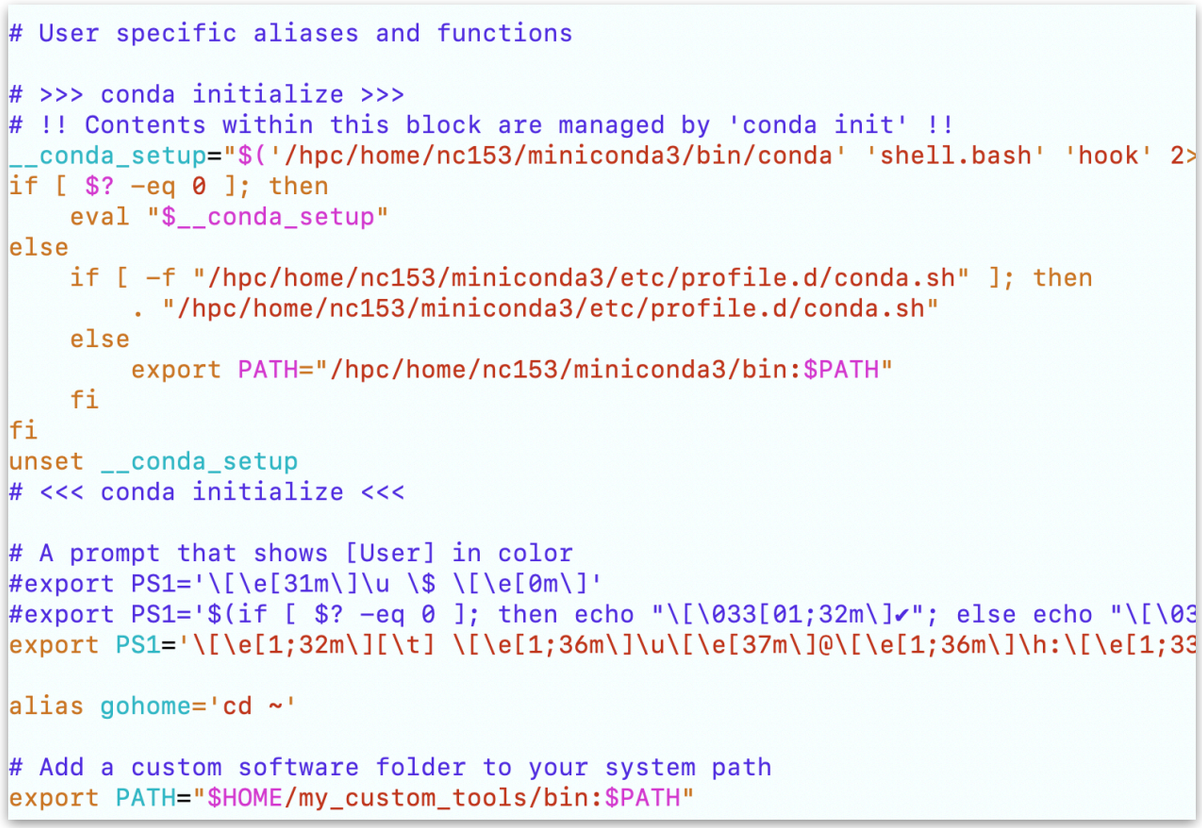
no-GUI

- Vi/Vim
- Emacs
- Nano
- …

Or...


File Edit View Run Kernel Tabs Settings Help
Launcher
jupyterlab.md
JupyterLab Demo
JupyterLab: The next generation user interface for Project Jupyter
https://github.com/jupyter/jupyterlab
It started as a collaboration between:
* Project Jupyter
* Bloomberg
* (then) Continuum
and now involves many other people from many other places (not pure
1) Building blocks of interactive computing
Start with the launcher
Use it to open different activities:

Vs.


# User specific aliases and functions
# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/hpc/home/nc153/miniconda3/bin/conda' 'shell.bash' 'hook' 2>
if [$? -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 "\[\033[01;32m\]✔"; else echo "\[\03
export PS1='\[\e[1;32m\][\t] \[\e[1;36m\]\u\[\e[37m\]@\[\e[1;36m\]\h:\[\e[1;33
alias gohome='cd ~'
Add a custom software folder to your system path
export PATH="$HOME/my_custom_tools/bin:$PATH"

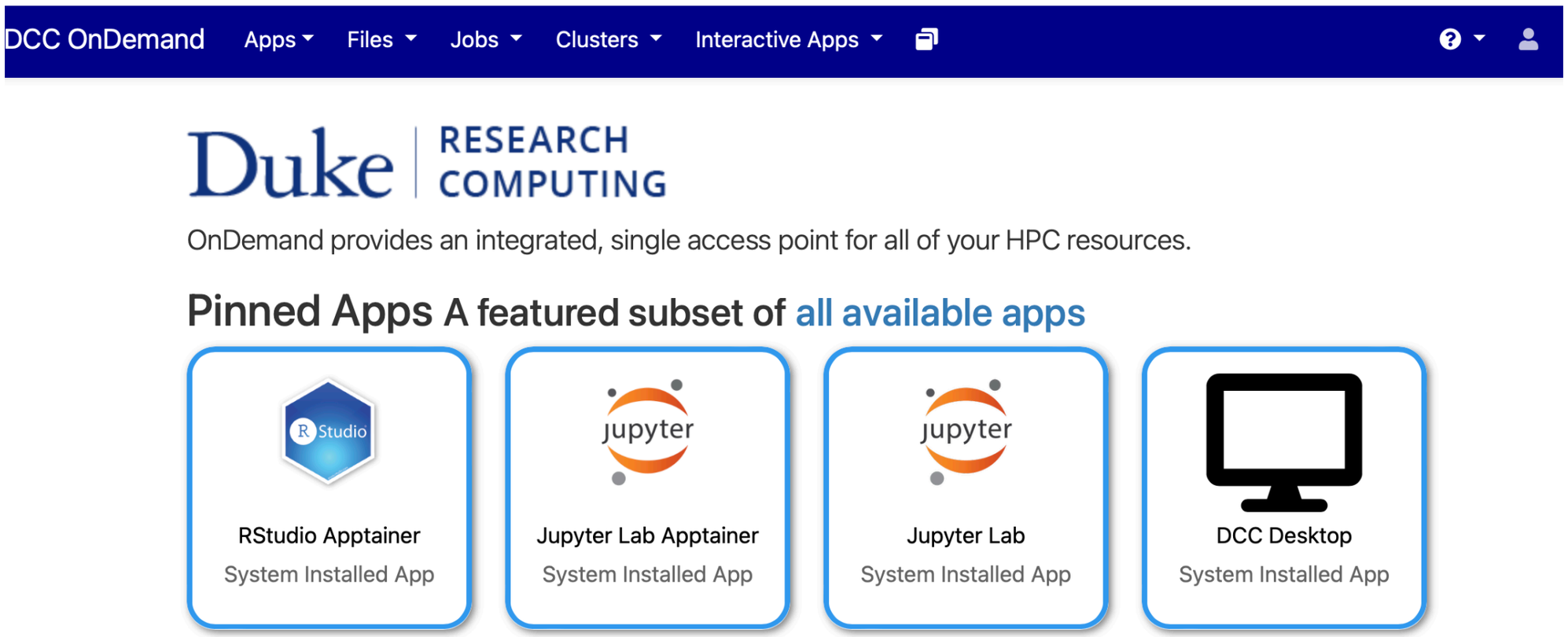
What are the disadvantages of GUI editors?

- 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



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

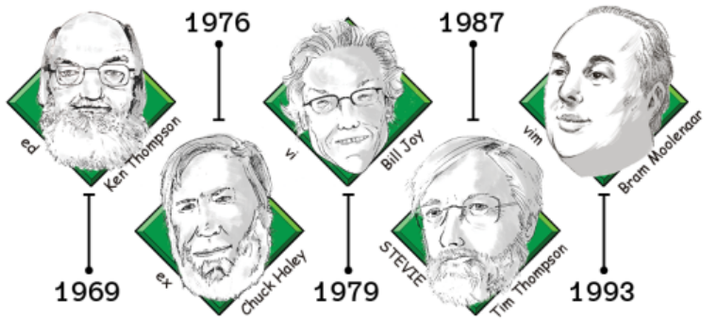
Inside a line

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

Inside a file

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

Basic Vim cheat sheet



Invocation

`vim -`	Read from STDIN
`vim +90`	Jump to line
`vim -u NONE`	“Debug” mode

sheet designed by Max Cantor thingsfittogether.com

Legend

- `a` Key press
- `a` ^a, <C-a>, `Ctrl` + a
- `a` Hold shift to modify usage

Registers “

`d d`	Cut line to default register
`p`	Paste from default register
`“ a p`	Paste from a
`“ a d d`	Cut line to a
`“ E 3 y`	Append 3 lines to e

Normal Mode i s a o c

`d`	
`y`	
`p`	
`g f`	Open file under cursor
`:`	Command Mode
`Z Z`	Save & Close
`Z Q`	Close
`6`	Swap files
`/`	
`n`	Next
`g g`	Word count
`o`	Jump back
`*`	Find word under cursor
`v`	Visual Mode

Insert Mode

`x` Complete...

- `f` File
- `l` Line
- `t` Tag

`n` Autocomplete

- `n` Next
- `p` Previous
- `y` Confirm
- `e` Cancel

`r “` Paste from “

`Esc` or `c` or `[` Return to Normal Mode

Options

`:set wrap lbr`	Notepad “mode”
`:set ft?`	Check filetype
`:set list et`	Spaces only
`:set ru nu sc`	Inspect “mode”
`:set sw=4 sts=4`	Tab-to-space ratio
`:set tw&`	Reset textwidth
`:set nohls`	Disable search hilighting

Command Mode

`Enter`	Submit command
`:quit`	Exit Vim
`:write`	Save file
`:pwd`	Current directory
`:ls`	List open files
`:cd`	Change directory
`:set`	Set options

`f` History

`:edit`	Open file
`:vsplit`	Vertical
`:close`	Close split
`:bn`	Go to file *n*
`:help`	Learn Vim
`:reg`	List all registers

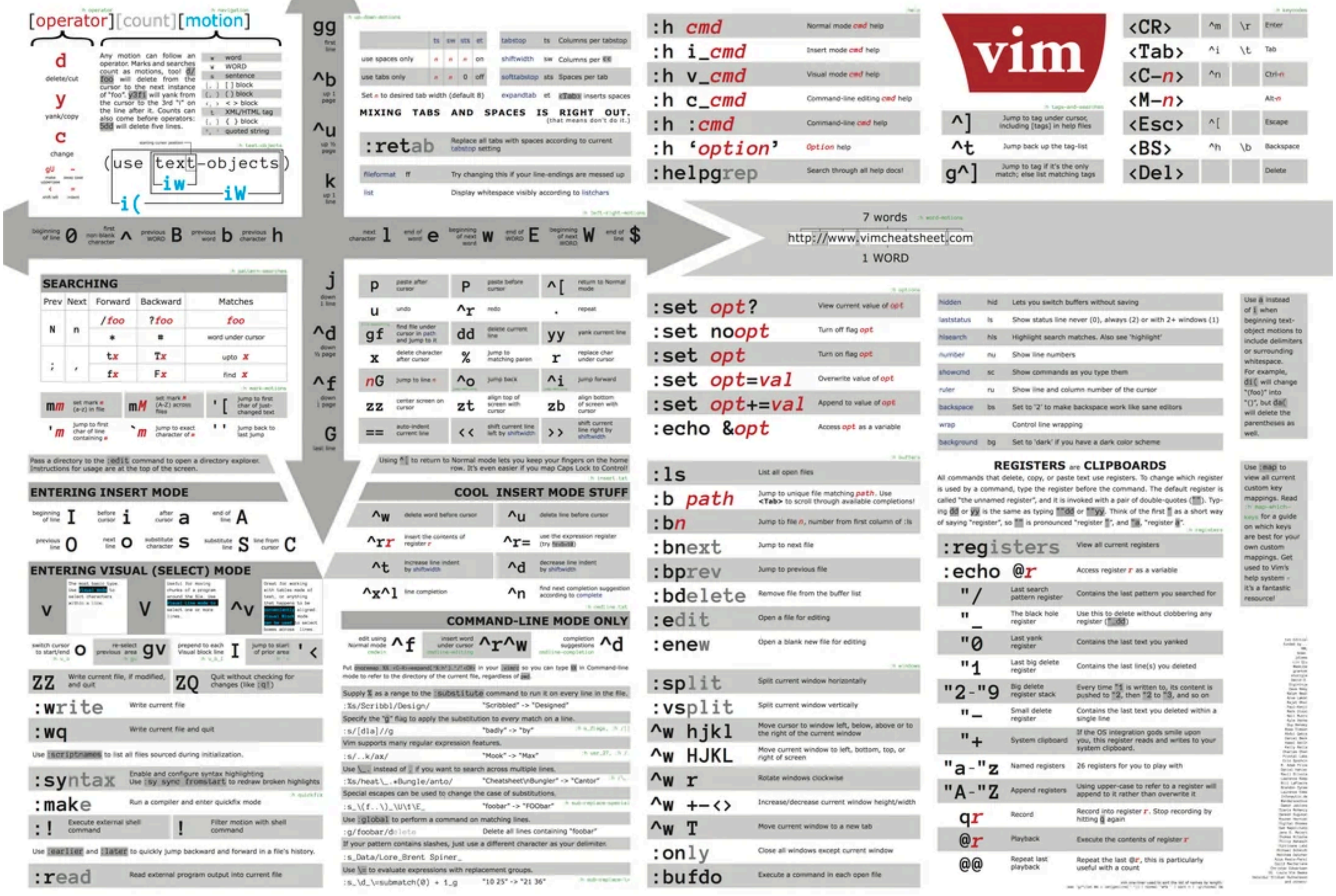
Manual ?

`]` Drill down `t` Bubble up

`:help navigation`	
`:help options`	
`:help windows`	
`:help motion`	
`:help registers`	
`:help cmdline`	
`:help :s`	Search & replace
`:help ^w`	`w`indow commands
`:help map`	Macros & keys
`:help i^w`	Insert mode `w`?
`:help word-motions`	
`:help pattern-searches`	

Source: https://thingsfittogether.com/

Advanced Vim cheat sheet



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

```
" --- General Settings ---
set nocompatible            " Use Vim defaults instead of old Vi behavior
syntax on                   " Enable color highlighting for code
set number                  " Show line numbers on the left
set cursorline              " Highlight the current line your cursor is on
set showcmd                 " Show the command you are typing in the bottom right

" --- Search Settings ---
set incsearch               " Highlight matches as you type your search
set hlsearch                " Keep matches highlighted after you hit Enter
set ignorecase              " Ignore case when searching...
set smartcase               " ...unless you use a capital letter

" --- Indentation & Tabs ---
```

```
#!/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: Medium complexity example

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

```
      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. PLUGINS
" ==============================================
call plug#begin('~/.vim/plugged')

" Aesthetics & UI
Plug 'joshdick/onedark.vim'              " Professional color scheme
Plug 'itchyny/lightline.vim'             " Lightweight status bar
Plug 'preservim/nerdtree'                " File explorer sidebar
Plug 'ryanoasis/vim-devicons'            " Icons for file types

" Navigation & Search
Plug 'junegunn/fzf', { 'do': { -> fzf#install() } }
Plug 'junegunn/fzf.vim'                  " Blazing fast fuzzy finder
Plug 'christoomey/vim-tmux-navigator'    " Seamless Vim/Tmux navigation

" Coding Intelligence
Plug 'neoclide/coc.nvim', {'branch': 'release'} " LSP support (IntelliSense)
Plug 'tpope/vim-commentary'              " Shortcut to comment code
Plug 'tpope/vim-fugitive'                " The best Git wrapper
Plug 'psf/black', { 'tag': 'stable' }    " Python formatter

call plug#end()

" ==============================================
" 2. CORE SETTINGS
" ==============================================
set nocompatible
filetype plugin indent on
syntax on
set termguicolors            " Enable 24-bit RGB colors
```

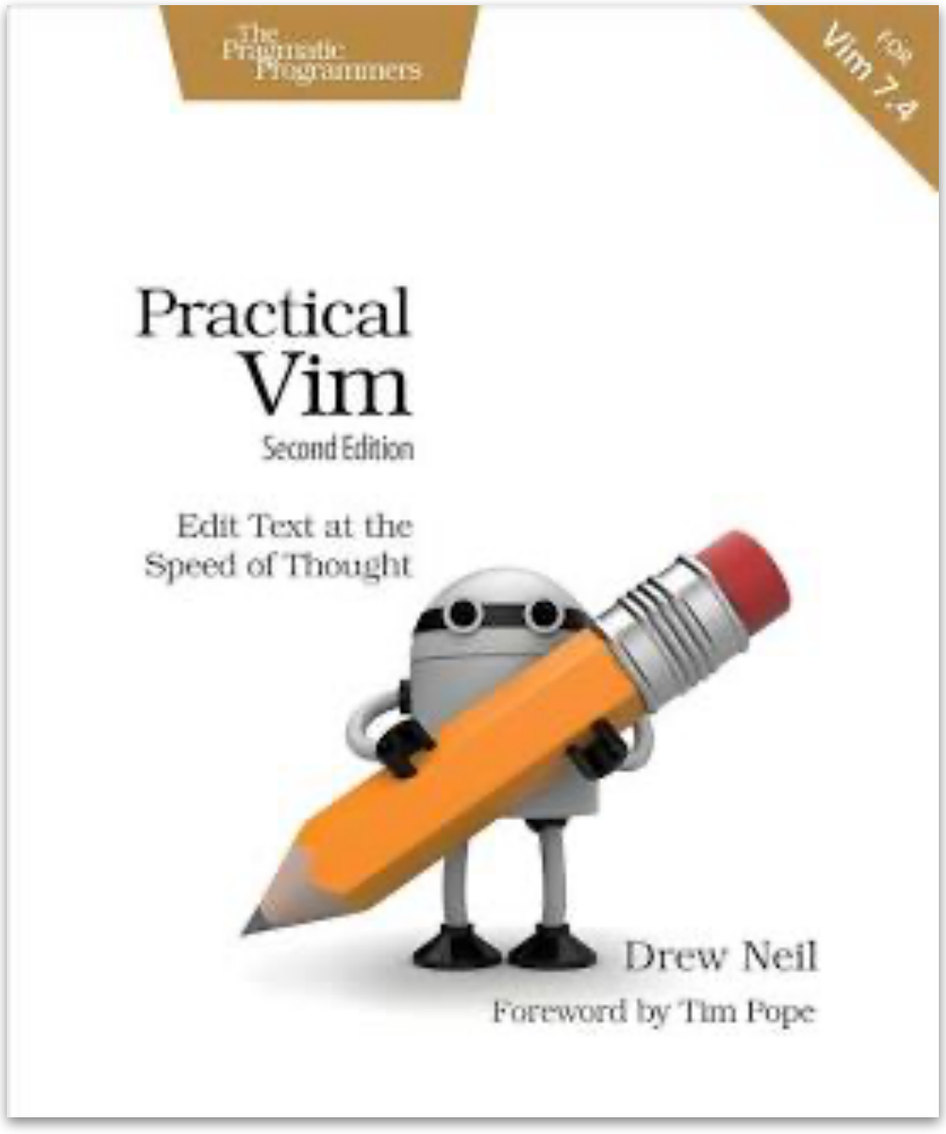
You will need to install plug-ins. I have never used this and didn’t know it was a thing until I made this lecture.

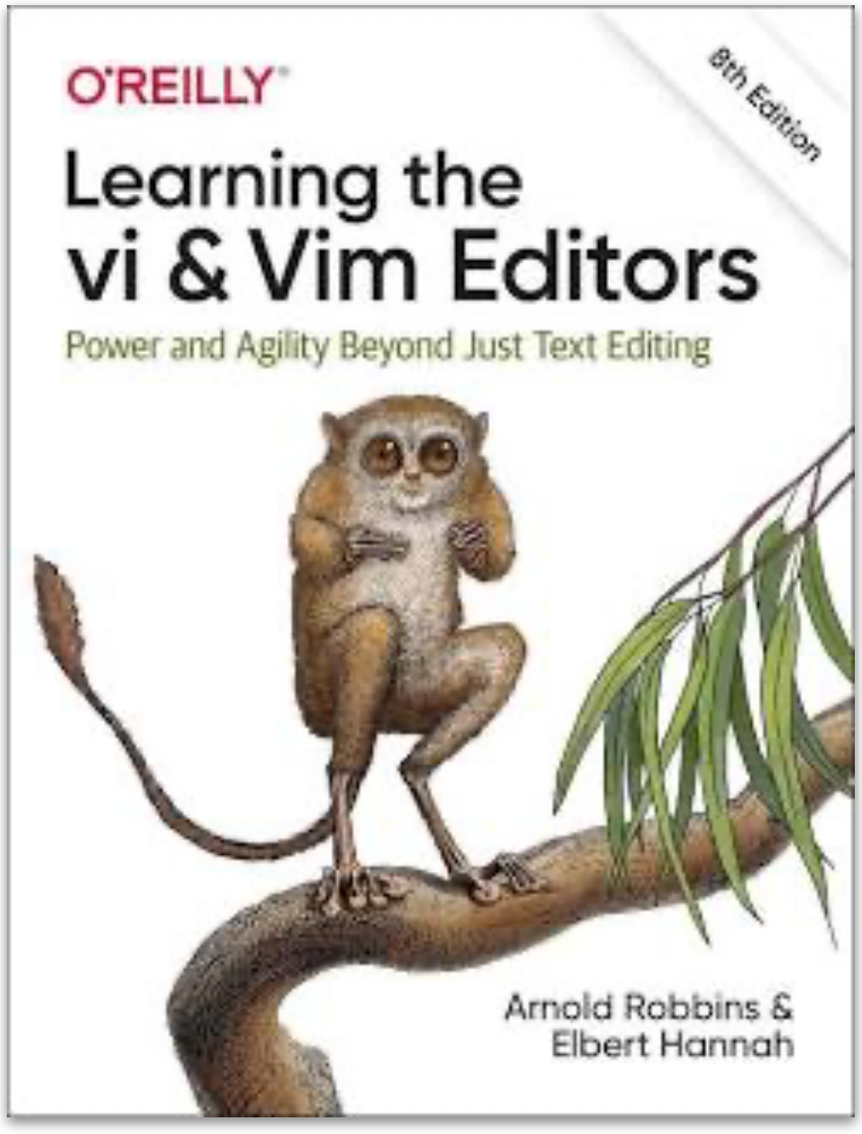
Vimtutor

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

```
===============================================================================
=    W e l c o m e   t o   t h e   V I M   T u t o r   -    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?


The Pragmatic Programmers
for Vim 7.4
Practical Vim
Second Edition
Edit Text at the Speed of Thought
Drew Neil
Foreword by Tim Pope


O'REILLY
8th Edition
Learning the vi & Vim Editors
Power and Agility Beyond Just Text Editing
Arnold Robbins & Elbert Hannah