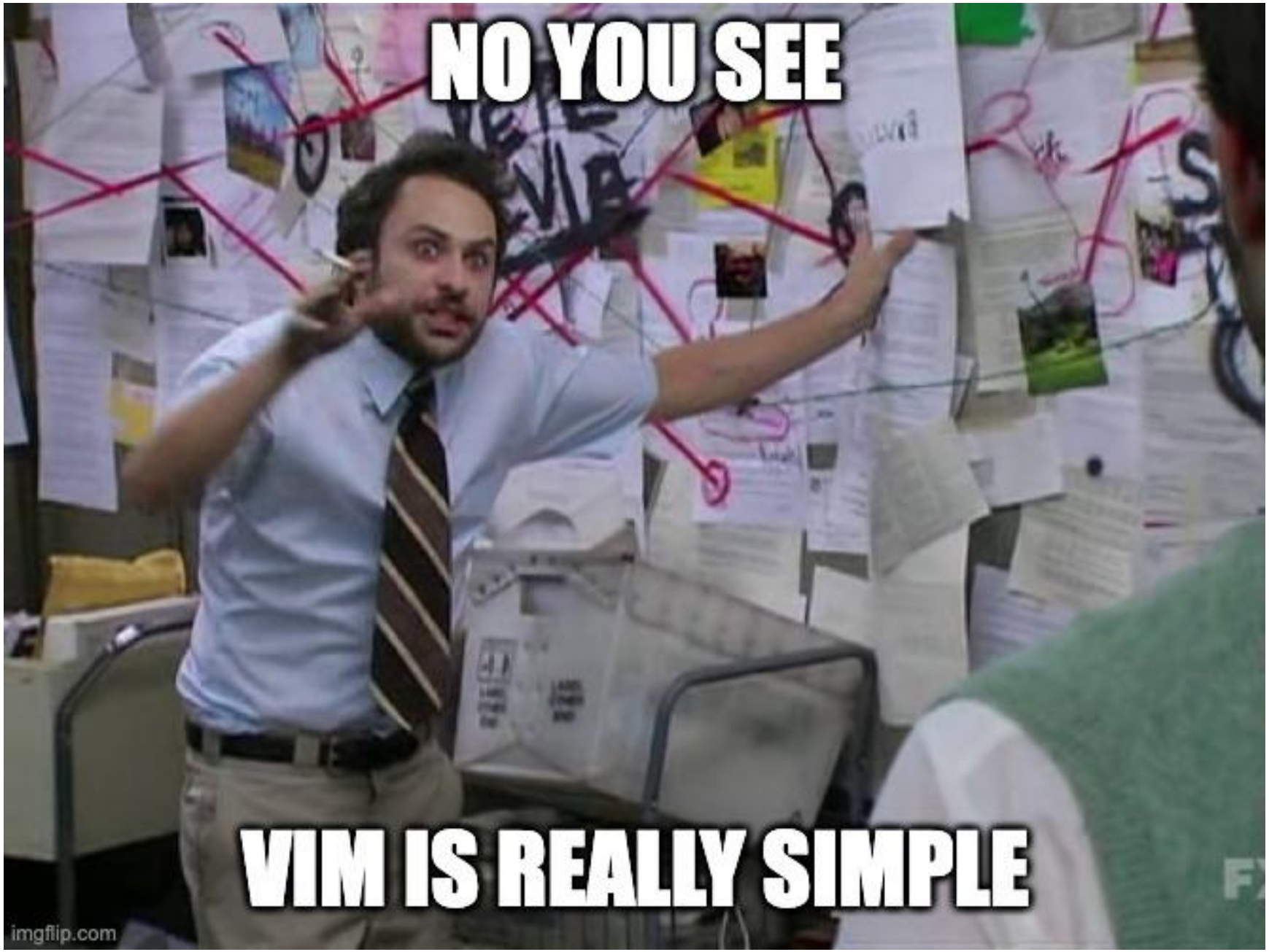
PERCONA

Everything you need to know about Vim and text on Linux

Joe Brockmeier
Head of Community @ Percona

What we'll cover



©2023 Percona | Confidential | Internal use only

The really big disclaimers!

- Just scraping the surface.
  - **Big** topic in a tiny timeslot.
- Based on current-ish versions of RHEL, Fedora, Debian, etc.
  - If you're using a more niche version of Linux the tools, defaults, etc. may vary.
- There's more than one way to do it.
  - Many of my examples are one way to do something, not the only way. Hopefully the best way to *illustrate* something but not necessarily the best way to do things in production.
- I'm a little biased. I'm very pro-Vim, but you might decide it's not for you. That's OK!

©2023 Percona | Confidential | Internal use only

Text basics

Why working with text on Linux matters

- It's 2023, do I still need this?
- STDIN, STDOUT, STDERR
- Redirecting input & output (text)
- OK, but why vi / Vim?

Vim necessities

The essentials and more

- What's a "modal" editor?
  - Normal mode
  - Insert mode
  - Command mode
  - Visual mode
- Getting in… and out of Vim
- Movement keys
- Operators + Motions
- Counts
- Undo + Redo
- Copy (yank) + Paste (put)
- Search + Replace
- Status (ctrl-g)
- Fancy features
- Customizing Vim

Text utilities

Other text utils that come in handy

- A person cannot live by Vim alone…
- GNU coreutils / textutils
  - head + tail
  - cat + tac
  - sort
  - uniq
  - cut
- Grep
- Pandoc is your friend

©2023 Percona | Confidential | Internal use only

Why do I still need this?

Working with text is a core skill on Linux

- Plain text is still the *lingua franca* of Linux.
- You still need to edit config files.
- Logfiles and output are usually in a text format.
- Modifying scripts, Dockerfiles, systemd units, and all manner of other operations require editing plain text files.

But… why vi / Vim?

- Vim is usually part of the default install on most Linux distributions – even minimal ones.
- Works well over SSH, no GUI required.
- It's *extremely powerful* once you really learn it.

©2023 Percona | Confidential | Internal use only

PERCONA

Text basics

Learning CLI tools in 2023

©2023 Percona | Confidential | Internal use only

Everything* is a file

- On Linux and UNIX type OSes, everything is a file.
  - Files are, of course, files.
  - Your devices are seen as files by the system, e.g. `/dev/ttyN` is a serial port (terminal), `/dev/pts/` devices are pseudo-terminals.
  - The `/proc` pseudo filesystem exposes information about the system:
    - `/proc/cpuinfo` is a "file" that tells you about your CPU, and `/proc/meminfo` tells you about your system's memory, *etc*.
- Why this matters:
  - Most of these files are text files and can be operated on or have information pulled from in plain text format.
  - Knowing how to work with text using CLI tools can make administration easier.

©2023 Percona | Confidential | Internal use only

stdin, stdout, stderr

- You have three “standard” streams on Linux to know about::
  - stdin: standard input – user input
    - this is fd 0 (`/dev/fd/0`)
  - stdout: standard output – command output
    - this is fd 1 (`/dev/fd/1`)
  - stderr: standard error – error messages
    - this is fd 2 (`/dev/fd/2`)
- These are all in text format.
- You can redirect their output!

©2023 Percona | Confidential | Internal use only

Redirecting input + output at the shell

- Command output, input, and errors can be redirected at the shell.
- Using < redirects to input
  - Example **`command <filename`** will send the contents of "**`filename`**" to "**`command`**" as if it were input.
- Using > redirects stdout
  - Example **`command >filename`** will send the output of "**`command`**" to "**`filename`**" rather than to the terminal.
- Using 2> redirects stderr
  - Example: **`command 2> filename`** will redirect error messages to "**`filename`**".
- Using > can write to a file (overwrite) – using >> will append.
- Pipe | will send content to another command.

©2023 Percona | Confidential | Internal use only

And now... Vim

A super-short history of vi / Vim

- The “vi” text editor was written by Bill Joy
  - vi stands for “visual” – it’s the “visual” mode for the even older ex editor
  - It dates back to 1976 and was not originally open source
- Vim is one of many clones of vi that extends functionality well beyond that of vi
  - Vim’s author and primary maintainer is Bram Moolenaar
  - Vim is “charityware” – the Vim.org site claims that its license is GPL-compatible but the license includes request to donate to the ICCF Holland foundation
  - Initial release was in 1991
- Vim, and vi, are what’s known as “modal” editors – that is, it works in different “modes” where the keys do different things based on what mode the editor is in
  - This will (I hope) make more sense shortly
  - It’s super-powerful for touch-typists and people who don’t like taking their hands off the keyboard to use menus, *etc*.

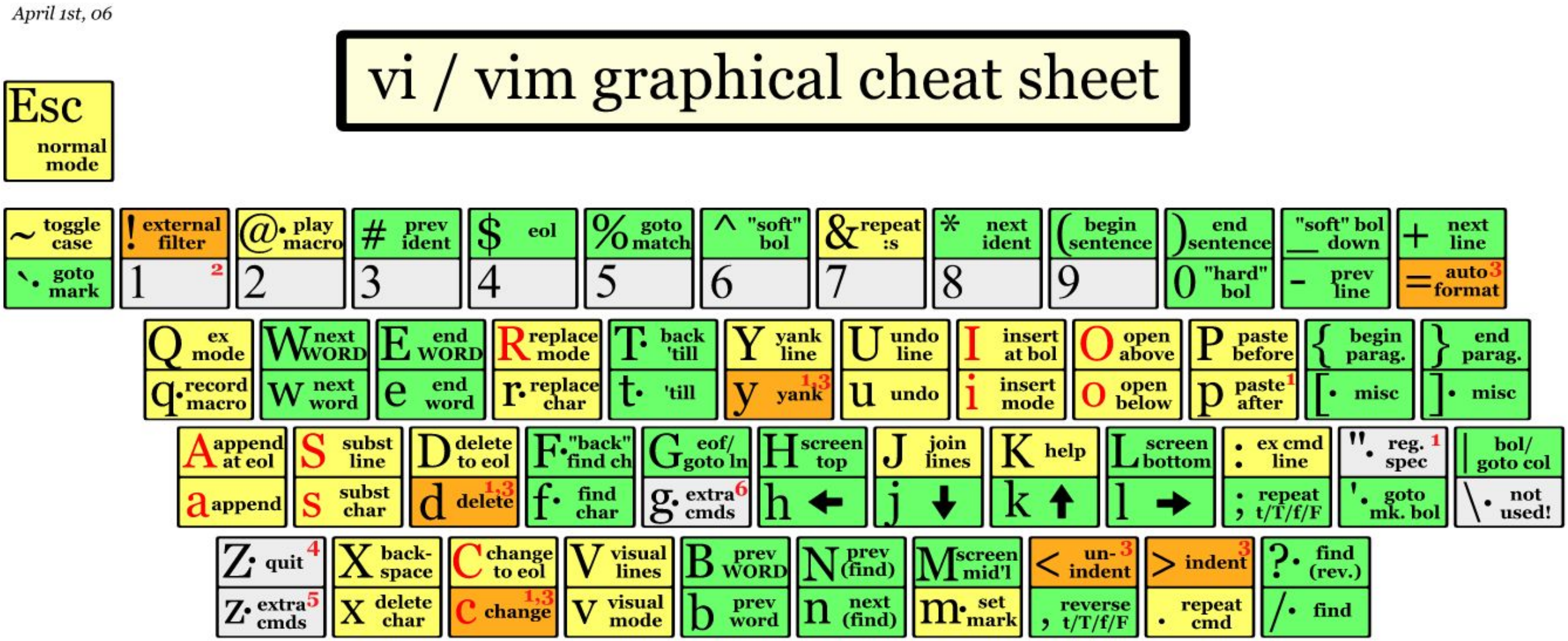
©2023 Percona | Confidential | Internal use only

The Vim 101: What's a "modal" editor?

- Depending on the mode you're in, key input is interpreted differently
- Modes to cover today:
  - **Normal mode** – in this mode, you move around and operate on the file. This includes operations like copy (yank) and paste (put), delete characters, append characters, etc. An "a" in normal mode puts you in "append" mode.
  - **Insert mode** – here you're writing (inserting) text. You type "a" and you get "a," etc. As you'd expect in any text editor or word processor.
  - **Visual mode** – this is a bit of a misnomer. Visual mode is used for selecting text.
  - **Command mode** – here you can issue longer commands like search and replace commands.
- There are additional modes but we'll skip over those for now.
- Like Linux and UNIX: Case matters!!

©2023 Percona | Confidential | Internal use only

The Vim 101: What's a "modal" editor?



Source: ViEmu: http://www.viemu.com/

©2023 Percona | Confidential | Internal use only

Modal tips

- Escape (**`Esc`**) and undo are your friends.
  - It's easy to fat-finger a keystroke and do something unintentionally.
  - Hit Esc and now you're in normal mode.
  - Hit **`u`** to undo the last change
  - Note the status at the bottom line when Vim undoes the action.

- If it has a **`:`** in front of it, it's a command mode operation.
- If you're writing, you're in insert mode.
- If you hit **`Esc`**, you're in normal mode.
- If you're selecting text, you're in visual mode.

©2023 Percona | Confidential | Internal use only

Starting Vim

- Open a terminal or log into the console - you should see a prompt like this: $
- Type vi or vim - you should see something like this:

```
~
~                          VIM - Vi IMproved
~
~                           version 8.1.3741
~                      by Bram Moolenaar et al.
~                 Modified by team+vim@tracker.debian.org
~                Vim is open source and freely distributable
~
~                       Become a registered Vim user!
~                type  :help register<Enter>   for information
~
~                type  :q<Enter>               to exit
~                type  :help<Enter>  or  <F1>  for on-line help
~                type  :help version8<Enter>   for version info
```

©2023 Percona | Confidential | Internal use only

Let's Edit! (and save)

When you start Vim you'll be in normal mode. You probably want to write some text at some point! Let's try insert mode.

- Type **`i`** to enter Insert mode from normal mode
  - If you're in any other mode, hit Escape (**`Esc`**) first
- Type some text – all of the alphanumeric keys work normally now.
- To return to normal mode, hit **`Esc`**
- **`Esc`** is your friend!
- To save type **`:w`** from normal mode (write)
- If you opened Vim without a filename, type **`:w filename`** and it will create + write the file!
  - You might want to choose a better name than "filename"...

©2023 Percona | Confidential | Internal use only

Exiting Vim: not obvious!

"There's more than one way to do it"

- Exit: (`:q`)
  - Enter command mode (:) and then type `q`
- Exit without saving changes: (`:q!` or `ZQ`)
  - Enter command mode (:) and then type q!
- Write the current file and then quit: (`:wq` or `ZZ`)
  - Enter command mode (:) and then type :wq

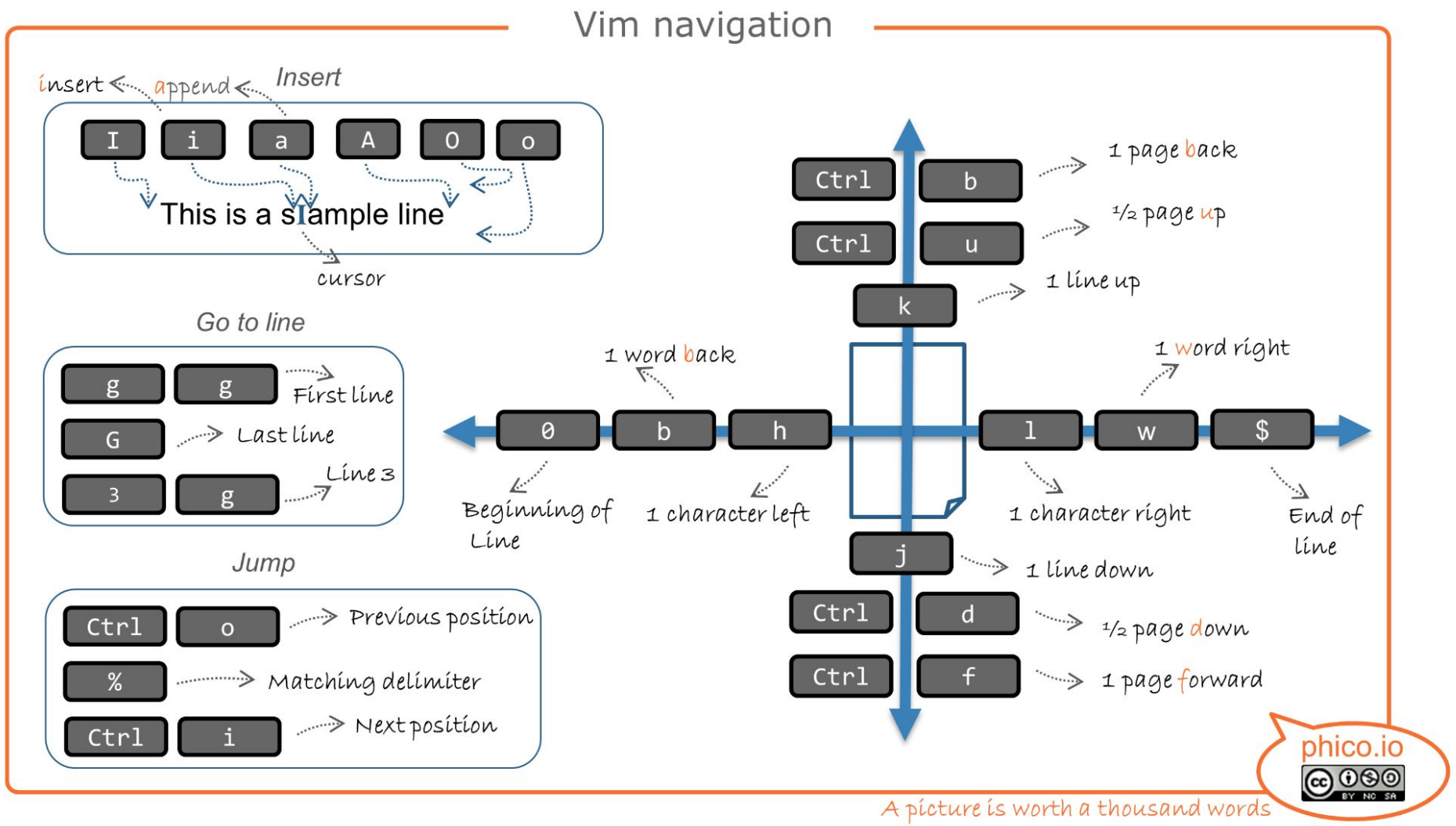
These are the basics – there are many other ways to quit if you have multiple buffers open.

©2023 Percona | Confidential | Internal use only

I have no mouse, but I must move…

- If your keyboard has arrow keys, they operate as you'd expect. (Up, Down, Left, Right)
- Hit Esc to enter "normal" mode
- The **`h j k l`** keys are your base movement keys.
  - **`h`** = left
  - **`j`** = down
  - **`k`** = up
  - **`l`** = right
- **`w`** will move you to the next string after a space
- **`b`** will move you to the beginning of the previous string
- **`H`** will take you to the top of the screen
- **`L`** will take you to the bottom of the screen
- **`$`** will take you to the end of a line
- **`0`** will take you to the beginning of a line
- **`G`** will take you to the end of the file
- **`gg`** will take you to the beginning of the file
- **`` `. ``** will take you to the position of the last edit

©2023 Percona | Confidential | Internal use only



Credit to phico.io!

©2023 Percona | Confidential | Internal use only

Undo, redo, and repeat

- Vim knows we're going to make mistakes, and it's happy to bail us out!
- Vim knows we do awesome work, and it's happy to help us repeat it!
- To undo the last operation (we won't say "mistake") type **`u`** in normal mode.
- To redo (undo the undo) the last operation type **`CTRL-r`** in normal mode.
- To repeat an operation you've just done use **`.`** in normal mode.
  - Example: If you enter insert mode and type "Vim is nifty" and then return to normal mode (**`Esc`**) and type **`.`** then Vim will repeat "Vim is nifty."
- These are usually modifiable. Let's talk about that next.

©2023 Percona | Confidential | Internal use only

Modifiers and operators

- Many operations in Vim will take a modifier. For example, you can modify many operations using a number.
  - `j` moves down one line
  - `5j` moves five lines down
  - `o` inserts a new line and drops you into insert mode
  - 5o does something really useful but maybe surprising!
    - As before o drops you into insert mode after inserting a new line.
    - You can type some text like “Vim is great!”
    - Hit Esc to return to normal mode.
    - Vim will repeat the same text 5 times!

©2023 Percona | Confidential | Internal use only

Buffers and Registers

- Let's talk about buffers and registers –
  - When Vim opens a file it creates a buffer – it doesn't make changes to the file itself until you tell it to do so!
  - Vim saves its changes in a swap file (such as .filename.swp) so that if Vim – or your whole system – crashes, you can recover your work.
  - Vim also has several "registers" where it can save copied text. We're not going to get deeply into this – but when you copy (yank) text, you can save it into a named register and maintain more than one at a time.
  - You can see all buffers with the `:ls` command
    - Note that Vim can have multiple windows and tabs open, and multiple viewports

©2023 Percona | Confidential | Internal use only

Editing multiple files with Vim / Viewports

- `:split` will let you split the viewport and see multiple views of the same file
- `:split filename` will open a new viewport with the file read into the buffer
- `:tabnew` will open a new tab with an empty buffer
- `:tabnew filename` will open a new tab with the file read into the buffer
- `:vsplit` does the same thing as split, but with a vertical split
  - I prefer `vsplit` when I'm working on a particularly wide monitor…
- When you have multiple files open in Vim you can operate on all of them or just one of them or just some… This can be handy if you want to search + replace text in a lot of files.

©2023 Percona | Confidential | Internal use only

Copy (yank) and paste (put)

- Vim refers to copying text as “yanking” text into a register. This might be too much detail except that the command for copy is “`y`” or “`Y`” – so remember “yank” for “copy.”
  - I like to think of it as “**yoink**” instead.
- Once you have text in a register you can *put* it into the file using “`p`” or “`P`”.
  - Lowercase “p” pastes text after the cursor.
  - Uppercase “P” pastes text before the cursor.

©2023 Percona | Confidential | Internal use only

Search and Replace

- **`/string`** will let you search forward through a file for **`string`**
- **`?string`** will let you search backward through a file for **`string`**
- **`*`** will search for the word the cursor is on
- **`n`** will take you to the next instance (if any)
- **`p`** will take you to the previous instance (if any)
  - Vim will say "**`search hit TOP, continuing at BOTTOM`**" or similar when you've looped around
- **`:s/string/newstring/`** will replace string on its current line.
- **`:s/string/newstring/g`** will replace **`string`** *globally*.
- **`:s/string/newstring/gc`** will confirm before replacing **`string`** *globally*.
- Use **`\`** to escape the **`/`** character if you want to search and replace it.
  - If you wanted to replace "**`/etc/filename`**" in a doc with "**`/etc/newfile`**" you'd use **`:s/\/etc\/filename/\/etc\/newfile/gc`**

©2023 Percona | Confidential | Internal use only

Deleting text

- **`x`** will delete a character under the cursor
- **`d`** can be used to delete larger selections of text
  - **`d3l`** will delete 3 characters to the right
    - Note that we're combining an operation, a movement, and a count here
  - **`d4h`** will delete 4 characters to the left
  - **`dd`** will delete a single line
  - **`d$`** will delete from the cursor to the end of the line
  - **`d0`** will delete from the cursor to the start of the line
- **`:7,10d`** will delete lines 7 to 10 in command mode
  - **`7,10`** is a "range"

©2023 Percona | Confidential | Internal use only

Working with ranges of text

- Vim is always thinking about the position of text, lines and columns
- **`:set number`** will turn on line numbering (or turn it off with **`:set nonumber`**)
- In command mode you can specify a range of text to operate on, e.g.:
  - **`:5,10d`** will delete the 5th through 10th lines.
  - **`:3,$d`** will delete everything from the 3rd line to end of buffer.
  - **`:1,4w filename`** will write lines 1-4 to “filename” if it doesn’t exist.
  - **`:1,4w! filename`** will write lines 1-4 to “filename” if it doesn’t exist, or overwrite the contents if it does.

©2023 Percona | Confidential | Internal use only

Getting status in Vim

- Vim displays status and errors at the bottom of the screen.
- `:f` will show you the file status including position.
- `g Ctrl-g` shows the exact position of the cursor by line, column, word, byte…
- `:ls` lists files in Vim

©2023 Percona | Confidential | Internal use only

Selecting Text in Vim

- **`v`** will set Vim to “visual” mode so you can select text.
- **`V`** will do the same, but select by line rather than by character.
- **`Ctrl-v`** will do the same but by column and line – this is useful for selecting columns of text to operate on!
- You can also use search in visual mode:
  - e.g. type “**`v`**” to enter visual mode and then **`/string`** to search for string.
  - Vim will select all the text from the cursor to the match for “**`string`**”.
- Once you’ve selected text you can operate on it in command mode – hit : to enter command mode and you’ll see `:'<,'>` as the range.

©2023 Percona | Confidential | Internal use only

Fancy features + customizations

- Vim has many, many, many more fancy features such as:
  - **gVim** - GUI for vim
  - **Easy mode** – only have vim but don't want to learn modes? Start with the `-y` option. Vim will behave a lot like a standard text editor with `Ctrl-s` for save, `Ctrl-c` for copy…
  - **External commands**: You can drop to a shell with `:sh` or execute a command with `:!` directly from Vim. You can also filter text through external commands using `!`
  - **Mappings**: Vim allows you to map your own keybindings to run complex commands.
  - **Abbreviations**: Set up abbreviations to avoid typing common phrases.
    - Used to use this for proper names that were lengthy or weird to type out like "openSUSE."
  - See: https://dissociatedpress.net/tag/vim/

©2023 Percona | Confidential | Internal use only

How to customize Vim

- **`:set [option]`** is used to set options for the session – they're not persistent
- To customize Vim on your system edit your **`.vimrc`**
  - Depending on the distro you may not have a **`.vimrc`** yet.
- Vim also has scripts and plugins!
  - See Vim.org and Vim Awesome for a ton of useful addons to Vim.

©2023 Percona | Confidential | Internal use only

Beyond Vim

Users cannot live on Vim alone...

Beyond Vim

- Linux is rich with utilities to manipulate text in ways that are useful for admins, programmers, and users.
- Quick reminder that it's a good idea to make copies of files or do run-throughs before doing operations that can be destructive!
- These tools are useful individually but very powerful when used together. Most are meant to do one thing very well.

©2023 Percona | Confidential | Internal use only

Text utilities

- **`head`** and **`tail`** – these utils will display (or pipe) the head or tail of files.
  - **`head -n 15`** will display the first 15 lines of a file.
  - **`tail -n 200`** will display the last 200 lines of a file.
  - Want line 35 from a file with hundreds of lines? **`head -n 35 filename | tail -n 1`**
  - **`tail -f filename`** will "follow" file – can be useful for logfiles where you're trying to watch information.

©2023 Percona | Confidential | Internal use only

cat and tac

- **`cat`** and **`tac`** – the **`cat`** util will "concatenate" and print a file or files to stdout. The **`tac`** command will do the same thing, backwards.
  - **`cat filename`** sends a file to the terminal.
  - **`cat filename1 filename2 > filename3`** will create **`filename3`** out of 1 and 2.
  - **`cat -n filename`** will output the file with line numbers.
  - **`gzcat filename`** will operate on gzipped files the same way cat works on plain text.

©2023 Percona | Confidential | Internal use only

sort and uniq

- **`sort`** does what you'd expect – it takes input and sorts it by (default) alpha order.
- **`uniq`** removes duplicate lines.
  - **`sort filename | uniq`** will sort the output and then pipe through uniq to remove duplicates.
  - **`sort -u filename`** does similar.
- There's significant overlap now where utils have options to do the same thing that other utils do. Sometimes the behavior is subtly different.

©2023 Percona | Confidential | Internal use only

cut

- **`cut`** will "cut" a file by specific parameters
  - **`cut -f 1,6 -d : /etc/passwd`** would print only the username and shell for users in **`/etc/passwd`**
    - **`d`** is the "delimeter" (`:`)
  - **`cut -c 1-20 filename`** would only print the characters in columns 1-20 for that file.
- This can be very useful to extract information from structured files like logs.

©2023 Percona | Confidential | Internal use only

grep

- **`grep`** will search files to match patterns. It's like google for your textfiles.
- Combine **`grep`** with other utilities for major superpowers.
  - **`grep Beatles albums.txt | sort`** would sort through "albums.txt" for lines with "Beatles" in them.
  - **`sudo tail -n 100 auth.log | grep Fail | grep sshd | grep root | cut -f 11 -d " " > banned-ip.txt`**
    would print out all the IP addresses trying to log into a system to a file called "**`banned-ip.txt`**"

©2023 Percona | Confidential | Internal use only

Pandoc is your friend

- Pandoc is worth a special mention. While not exactly like the others it's the Swiss Army Chainsaw of converting formats.
- You can use pandoc to convert from formats like HTML, Markdown, LaTeX and others into formats like Markdown, HTML, ePub, asciidoc and so forth.
- Why do I mention this? If you prefer writing in Vim (as I do) you can write in a format like Markdown for first drafts and then use Pandoc to convert to Word or other formats for sharing with colleagues.
  - **`pandoc -f markdown -o filename.docx original.md`**

©2023 Percona | Confidential | Internal use only

Thank You!