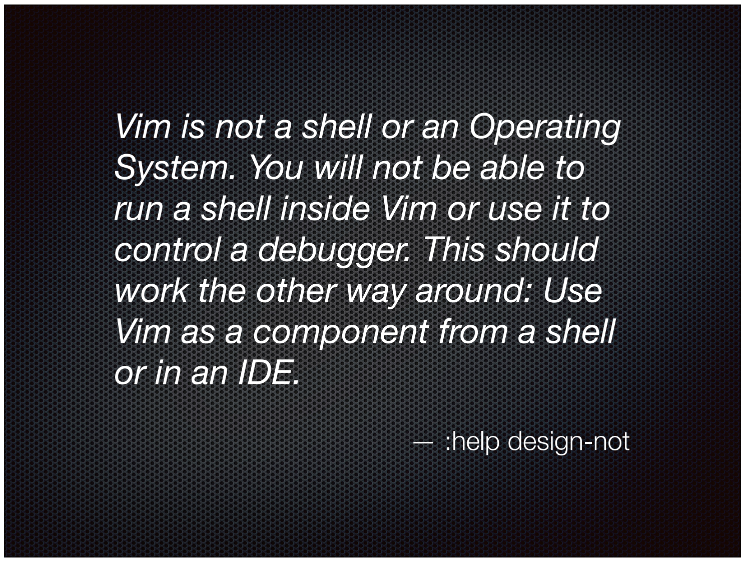
See Vim Run. Run, Vim, Run!

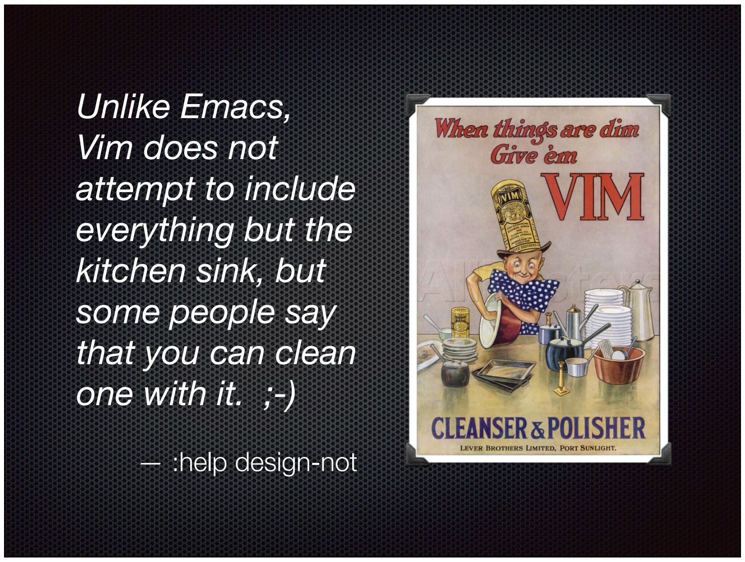
St. Louis Unix Users Group
Bill Odom · @wnodom · bill@billodom.com



Vim is not a shell or an Operating System. You will not be able to run a shell inside Vim or use it to control a debugger. This should work the other way around: Use Vim as a component from a shell or in an IDE.

— :help design-not

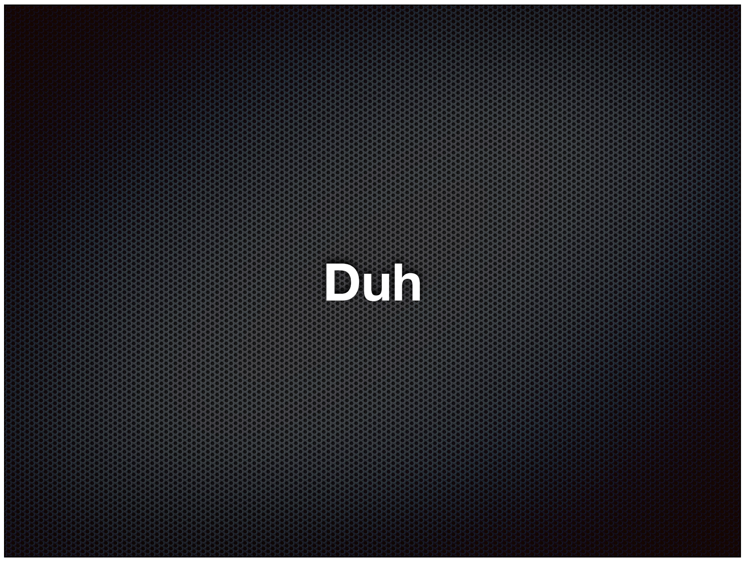
This isn't as true as it once was. Vim has grown to include many features that were once separate commands. There's a slide near the end that mentions several of them.



Unlike Emacs, Vim does not attempt to include everything but the kitchen sink, but some people say that you can clean one with it. ;-)

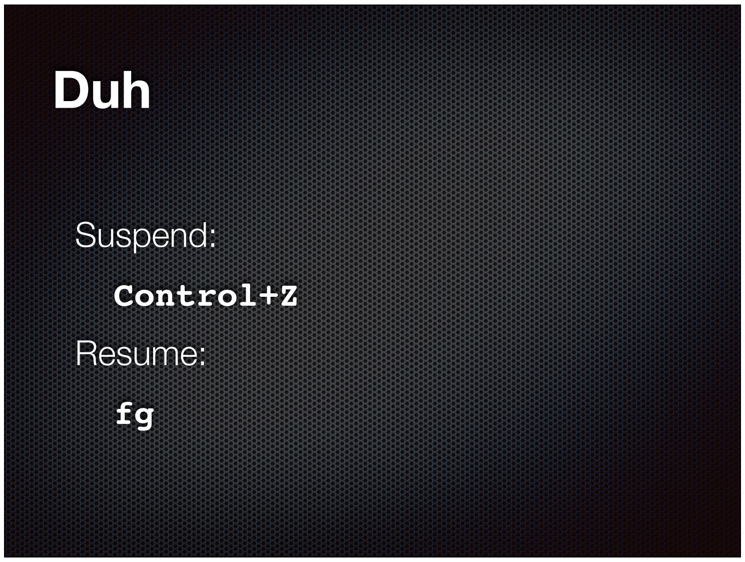
— :help design-not

It wouldn't be right to mention Emacs in a Vim presentation without including a photo of Saint IGNUcius. <https://stallman.org/saint.html>



Duh

The easy stuff you're likely to know already if you've been using Vim for even a short period of time.



Duh

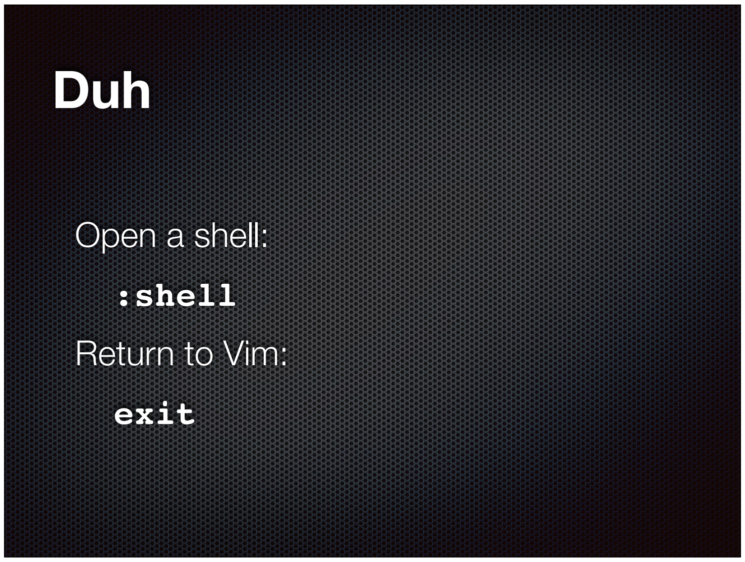
Suspend:

`Control+Z`

Resume:

`fg`

This mostly applies to terminal versions of Vim. In GUI Vim, Control +Z typically does whatever `:suspend` does, unless it's been mapped to something like undo.



Duh

Open a shell:

```
:shell
```

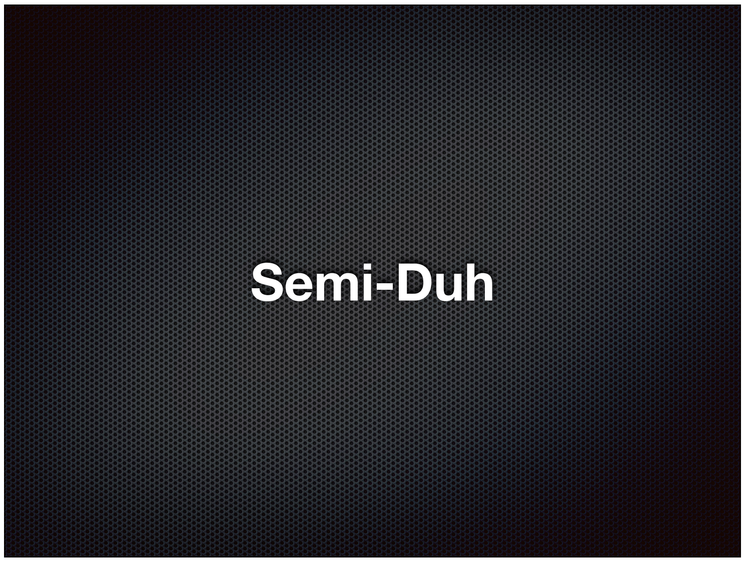
Return to Vim:

```
exit
```

`:shell` is often abbreviated as `:sh`

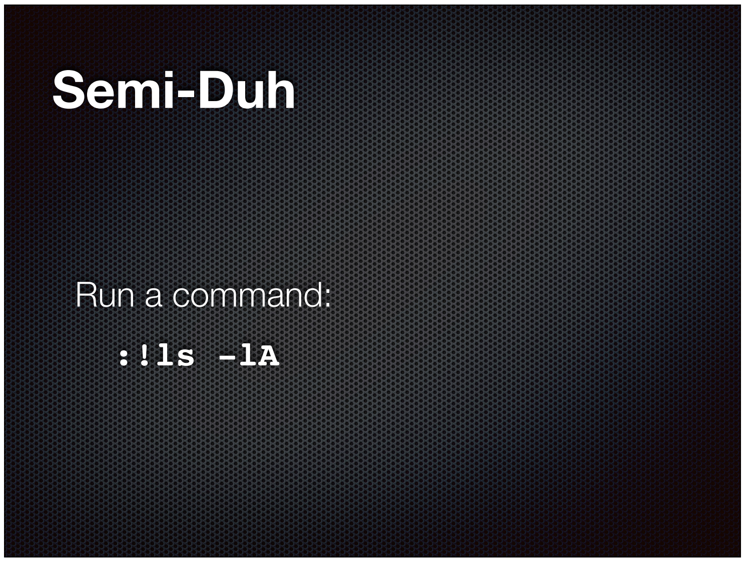
Check the `shell` setting to see what command Vim uses to start a shell: `set shell?`

Lots of other settings affect starting a shell (and therefore running an external command): `help 'shell'`



Semi-Duh

These are a little more complicated, but still pretty well-known.

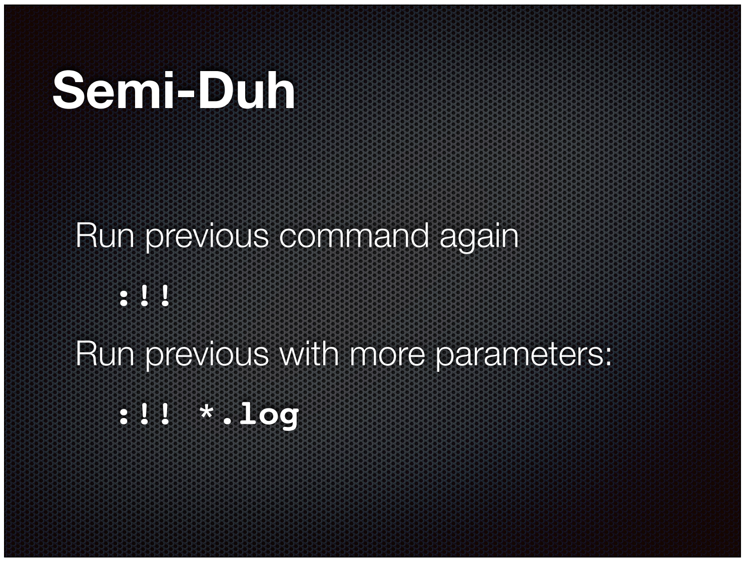


Semi-Duh

Run a command:

```
:!ls -lA
```

Just runs a command, without piping anything to it, and without replacing anything in the buffer.



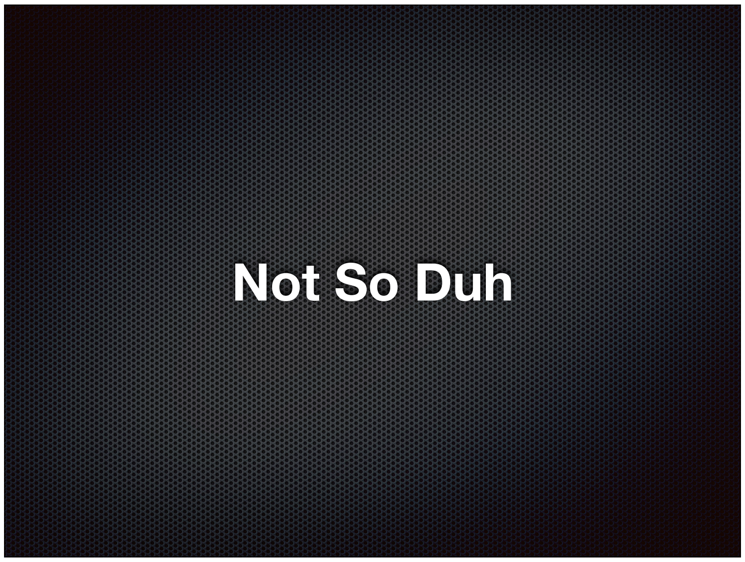
Semi-Duh

Run previous command again

```
:!!
```

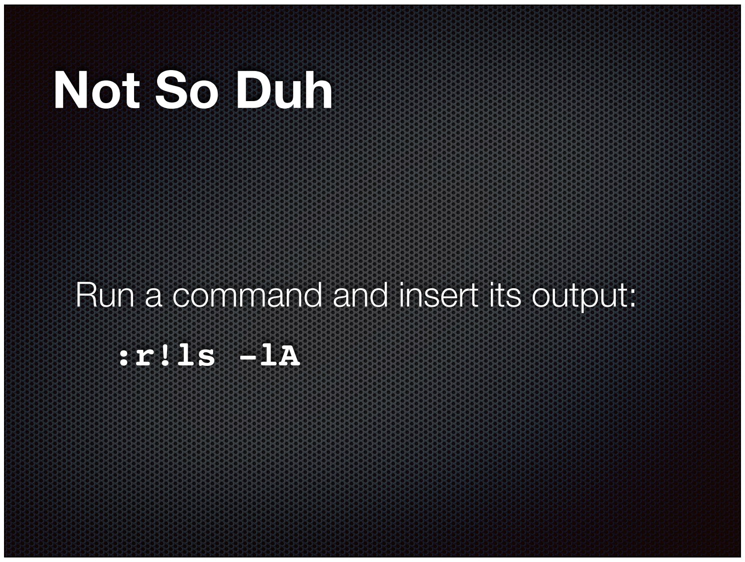
Run previous with more parameters:

```
:!! *.log
```



Not So Duh

Even experienced Vim users might be surprised by some of these.



Not So Duh

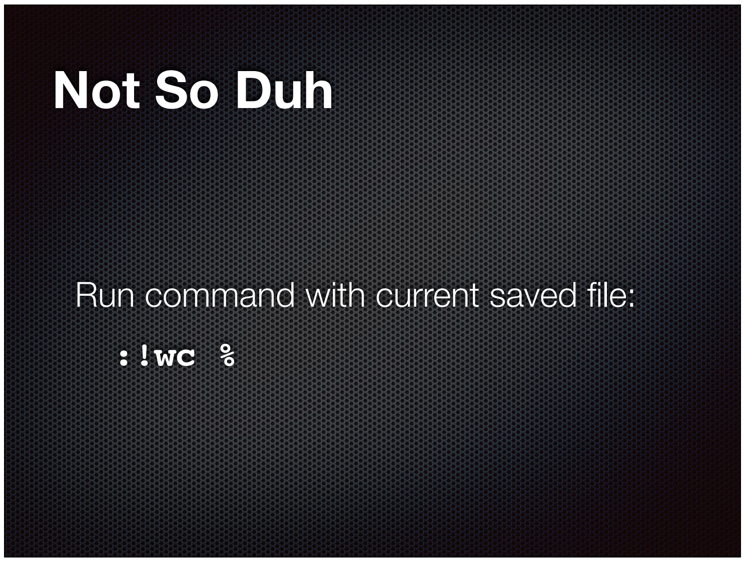
Run a command and insert its output:

```
:r!ls -lA
```

You can also use `:r` to just read a file into the buffer. You don’t have to run a command.

Some people like to include a space between the r and the !, but it’s not required (but see important note for `:w !…` later).

This works just as well with a pipeline.

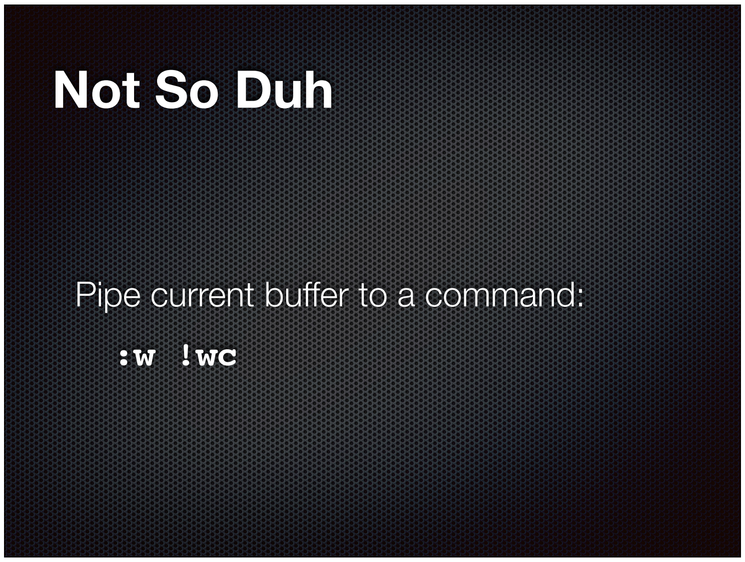

Not So Duh

Run command with current saved file:

```
:!wc %
```

The `%` expands to the filename associated with the current buffer.

See `:help cmdline-special` for several other special characters, and `:help filename-modifiers` for all the stuff you can do to them.

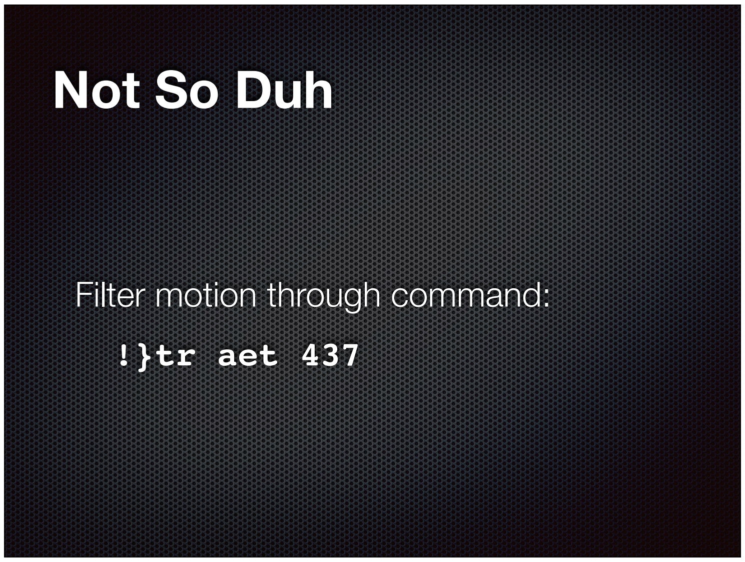

Not So Duh

Pipe current buffer to a command:

```
:w !wc
```

Note the space — leaving out the space between the w and the bang doesn't work the same way.

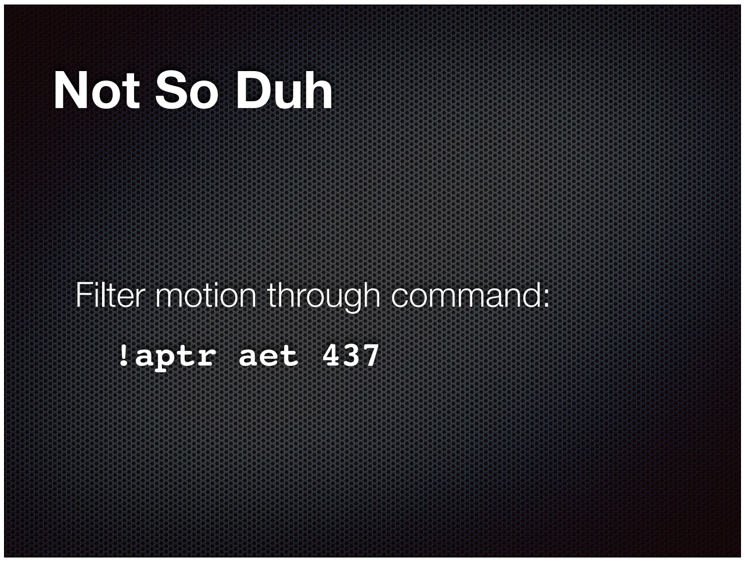
The buffer doesn't have to be saved to disk for this to work. In other words, the contents of the buffer and the actual file on disk (if there is one) aren't necessarily the same.


Not So Duh

Filter motion through command:

```
!}tr aet 437
```

Isn't that lovely? This filters text from the current line to the end of the current paragraph through the `tr` command, converting a, e, and t to 4, 3, and 7, respectively.



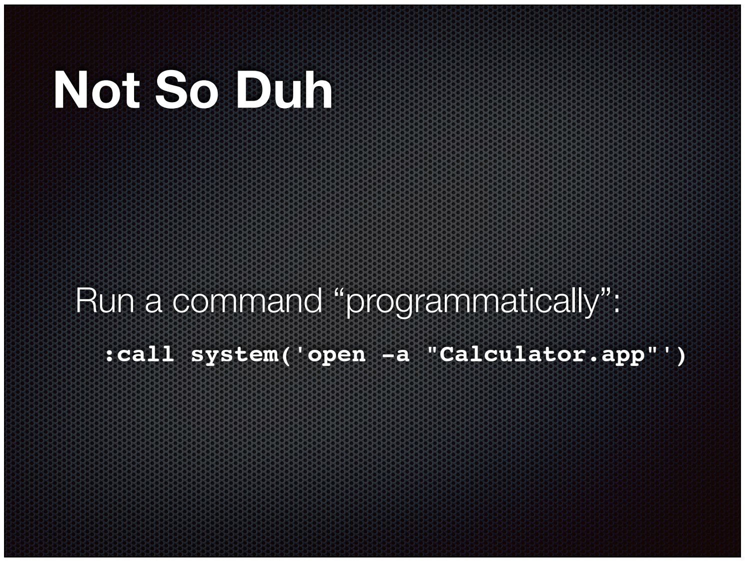
Not So Duh

Filter motion through command:

```
!aptr aet 437
```

`!` also works with text objects, by moving the cursor to the beginning of the object, then specifying a range.

The slide is kind of confusing because `!aptr` sort of looks like you're running a command called `aptr`. It's actually `!ap` ("filter a paragraph through the following command") followed by the `tr` command itself. It makes more sense as you're using it interactively.

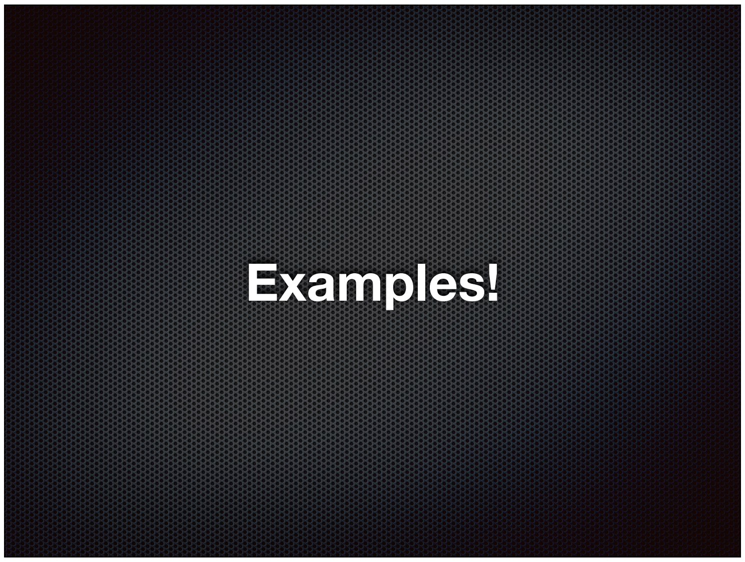


Not So Duh

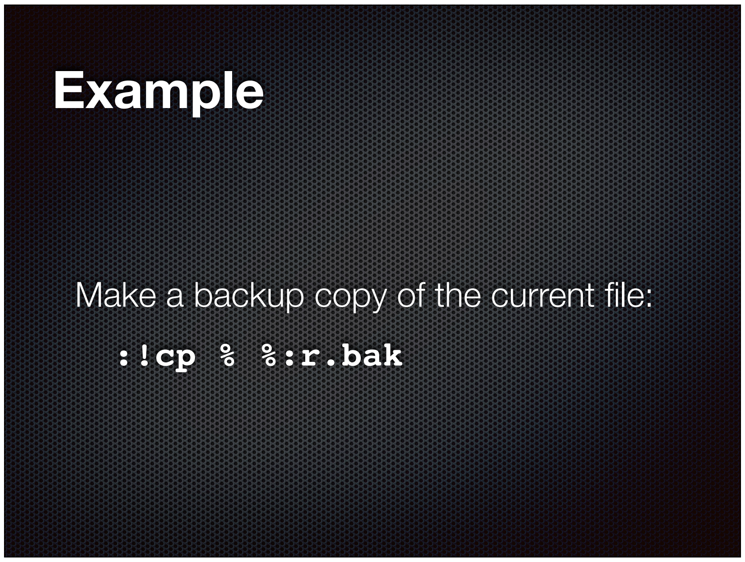
Run a command "programmatically":

```
:call system('open -a "Calculator.app"')
```

This isn't really intended for interactive use, but is very useful for scripting.



Examples!



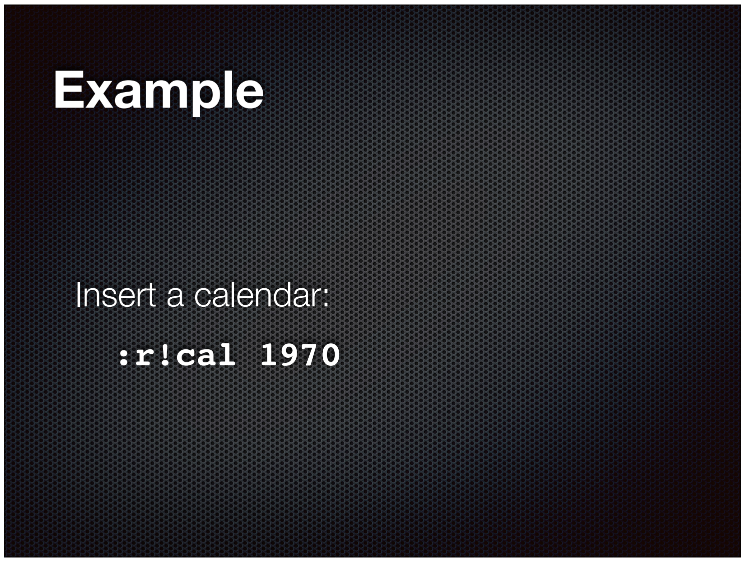
Example

Make a backup copy of the current file:

```
:!cp % %:r.bak
```

An example of using the “root” modifier with `%` — see`:help filename-modifiers` for more.

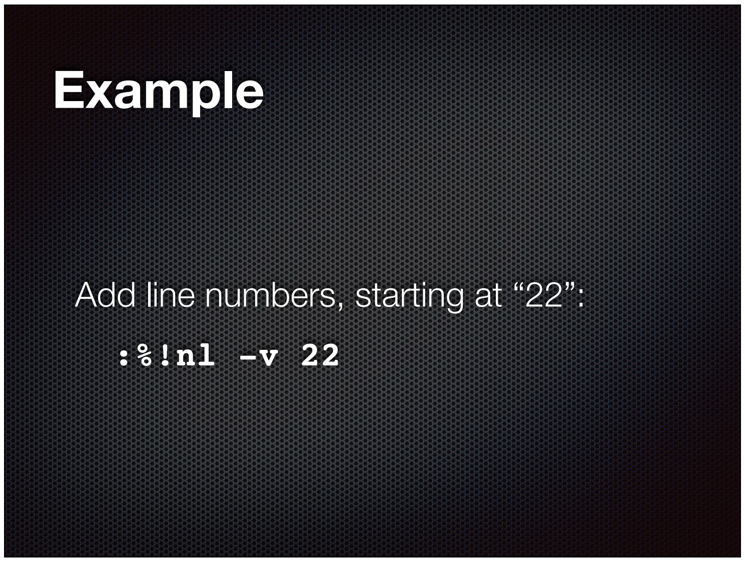
Be sure to quote or escape filenames that contain spaces or other special characters. If you need to do something like this in a script, consider using `system()` instead.



Example

Insert a calendar:

```
:r!cal 1970
```



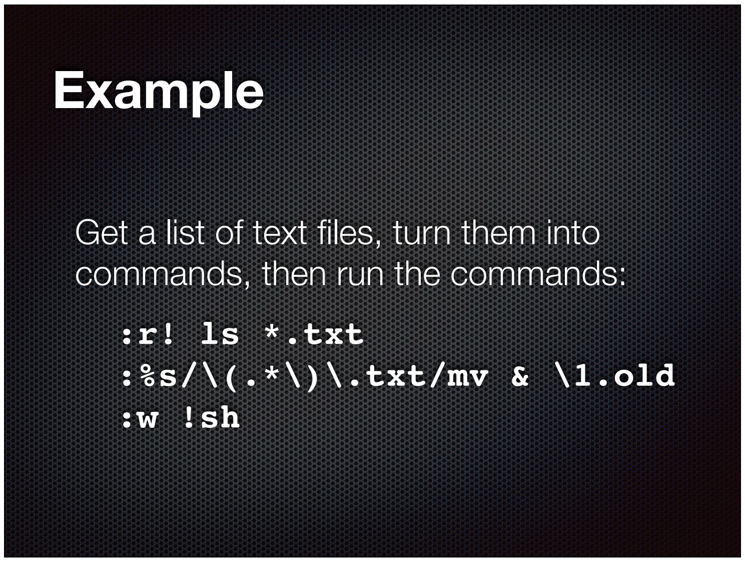
Example

Add line numbers, starting at “22”:

```
:%!nl -v 22
```

This is an example of using `%` as a range, *not* as the name of the current file. Similar concepts, but not the same thing. Be sure to understand the differences.

You can, of course, number lines in Vim itself with a little scripting, but this is quick and easy, especially if you already know how to use `nl`.



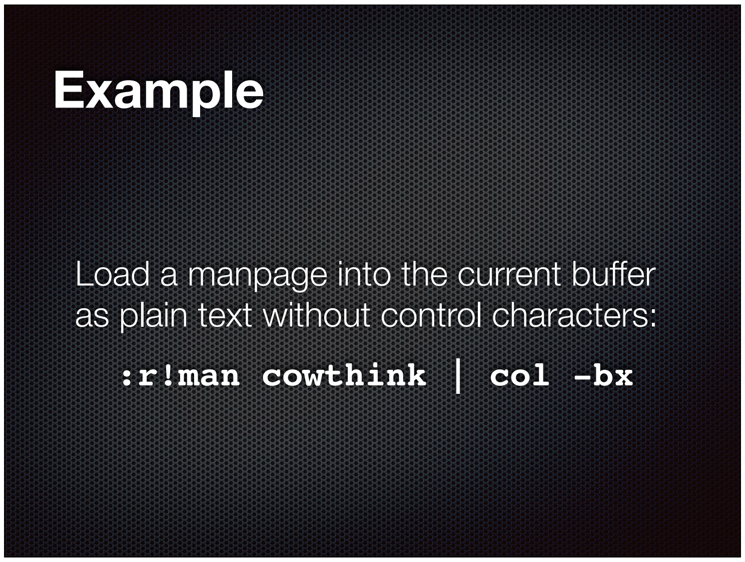
Example

Get a list of text files, turn them into commands, then run the commands:

```
:r! ls *.txt
:%s/\(.*\)\.txt/mv & \1.old
:w !sh
```

Note the use of `sh` as a filter.

Again, be careful about escaping / quoting filenames where necessary.



Example

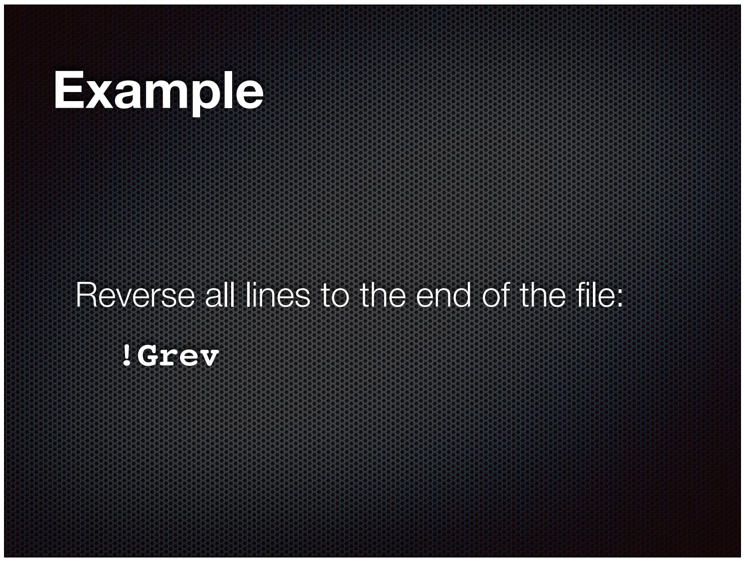
Load a manpage into the current buffer as plain text without control characters:

```
:r!man cowthink | col -bx
```

Conversion technique from <https://kb.iu.edu/d/acjn>.

Note how the `:r!...` technique works just as well with a pipeline as with a single command.

(This example assumes you have the cowsay package installed, which includes `cowthink`.)



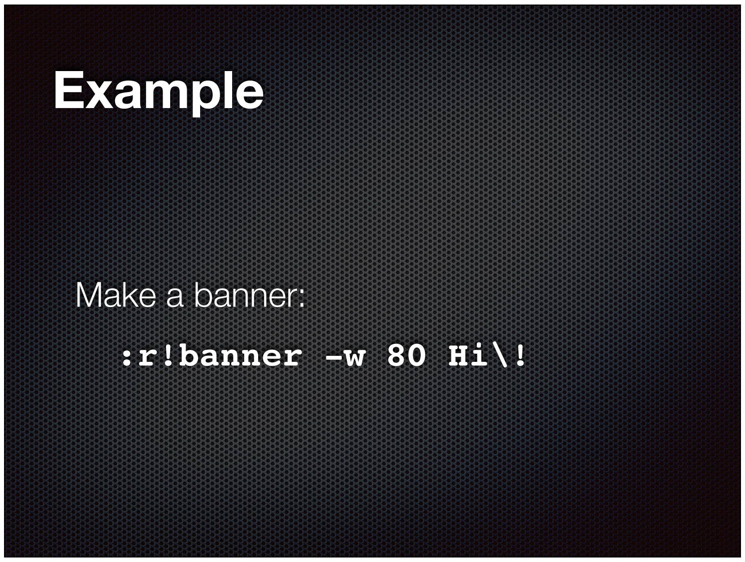
Example

Reverse all lines to the end of the file:

```
!Grev
```

Or, you know, you could do this…

```
    :%s/\(\<.\{-}\>\)/
\=join(reverse(split(submatch(1), '.
\zs')), '')/g
```

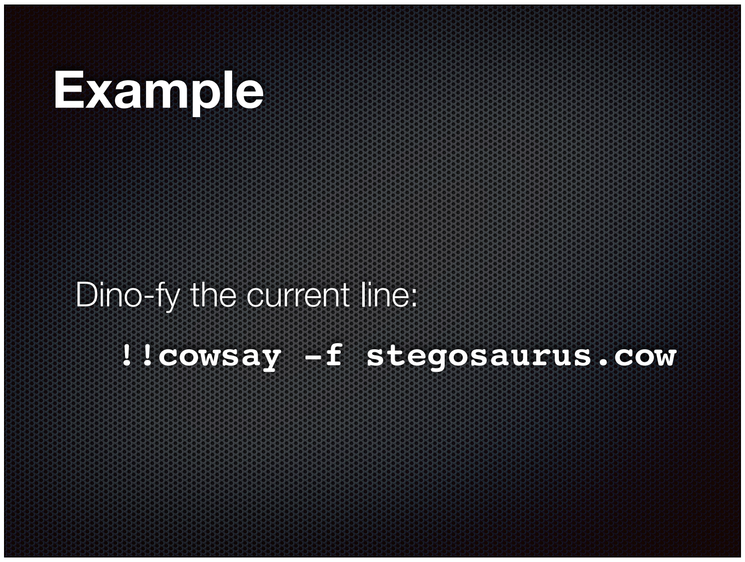


Example

Make a banner:

```
:r!banner -w 80 Hi\!
```

Note that `!` is escaped to keep it from being expanded to the text of the previous command.



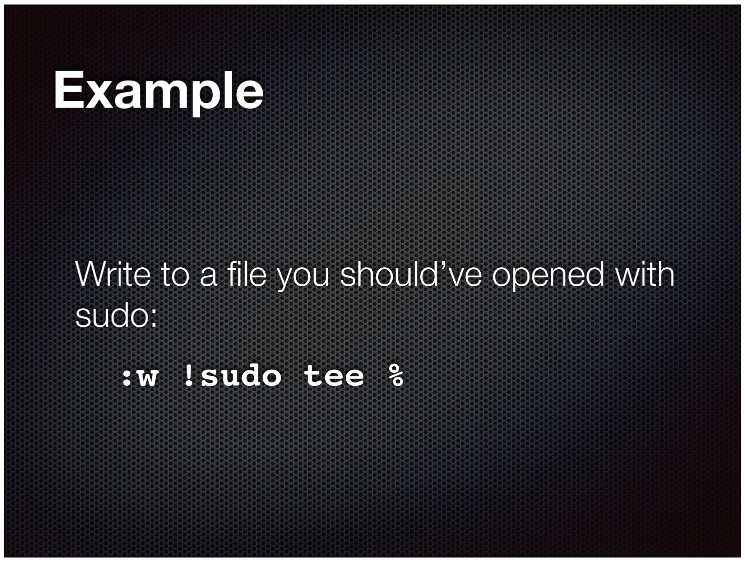
Example

Dino-fy the current line:

```
!!cowsay -f stegosaurus.cow
```

`!!` filters the current line through the specified command.

(This example assumes you have the cowsay package installed.)

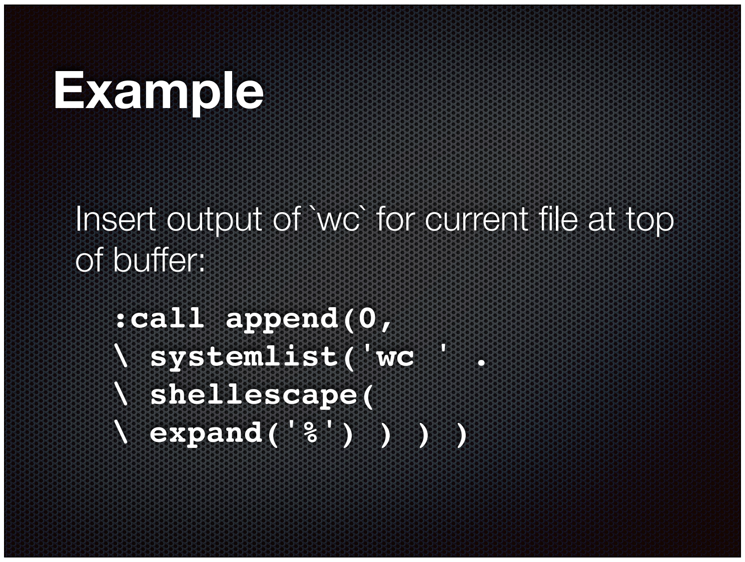


Example

Write to a file you should've opened with sudo:

```
:w !sudo tee %
```

This is a classic, but now you know how it works. :)



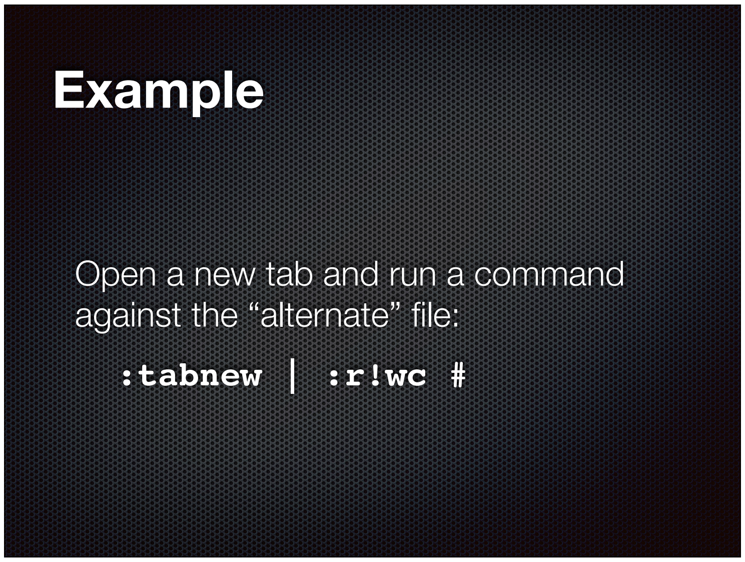
Example

Insert output of `wc` for current file at top of buffer:

```
:call append(0,
\ systemlist('wc ' .
\ shellescape(
\ expand('%') ) ) )
```

This is more like what you'd use in a script or plugin.

Broken across multiple lines, just to make it (slightly) clearer.

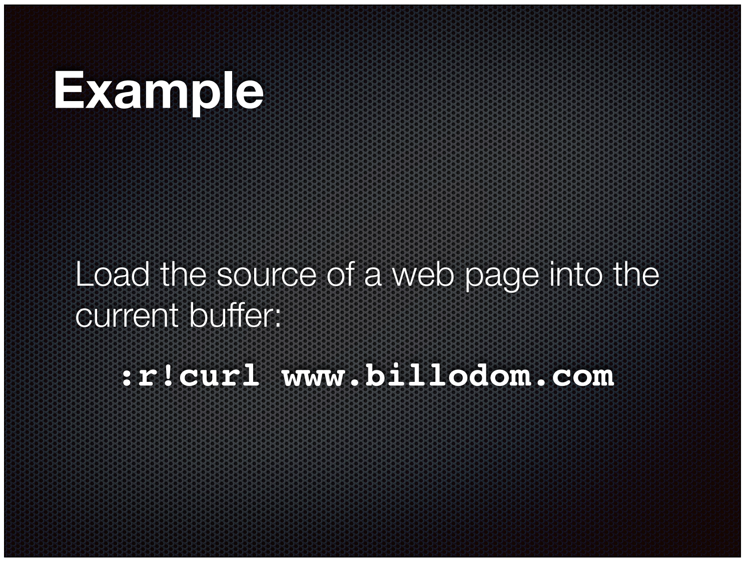


Example

Open a new tab and run a command against the "alternate" file:

```
:tabnew | :r!wc #
```

Sometimes you want to run a command against the current file, but in a new tab. As soon as you open a new tab, however, the "current" file isn't current anymore — it's become the "alternate" file. The `#` command-line special character lets you access the alternate file.

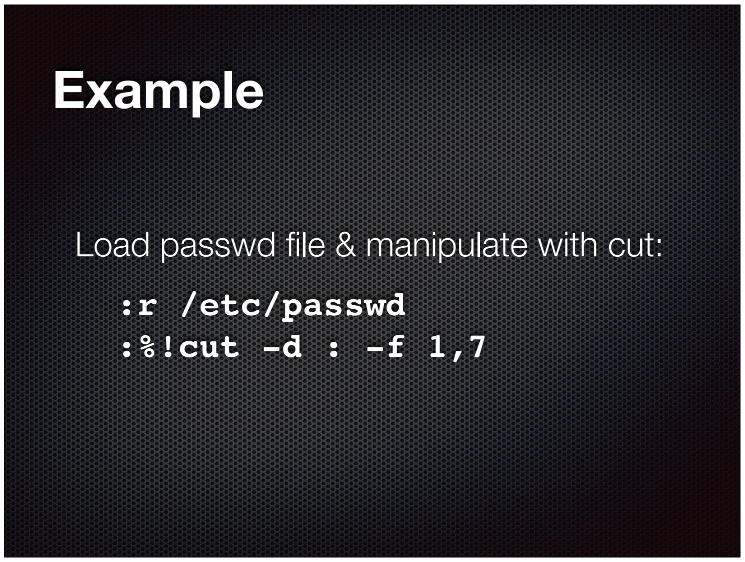


Example

Load the source of a web page into the current buffer:

```
:r!curl www.billodom.com
```

Nothing special about `curl`; could also use wget, lynx, etc.

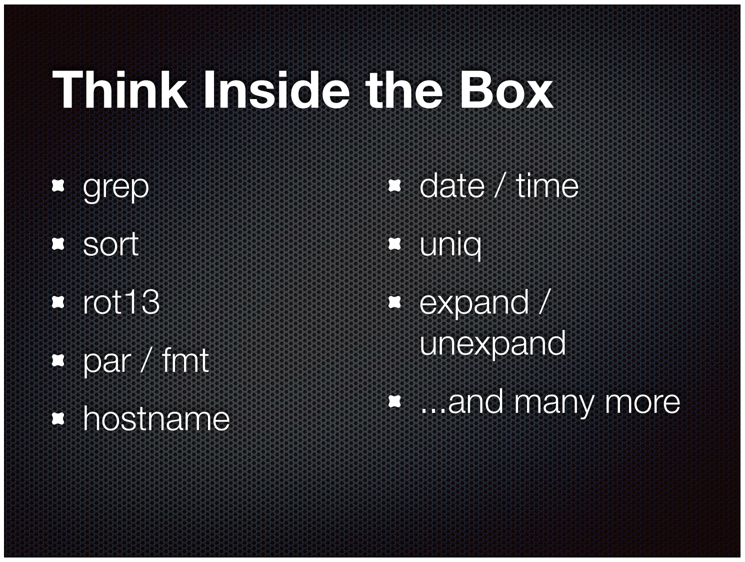

Example

Load passwd file & manipulate with cut:

```
:r /etc/passwd
:%!cut -d : -f 1,7
```

Or do both at once:

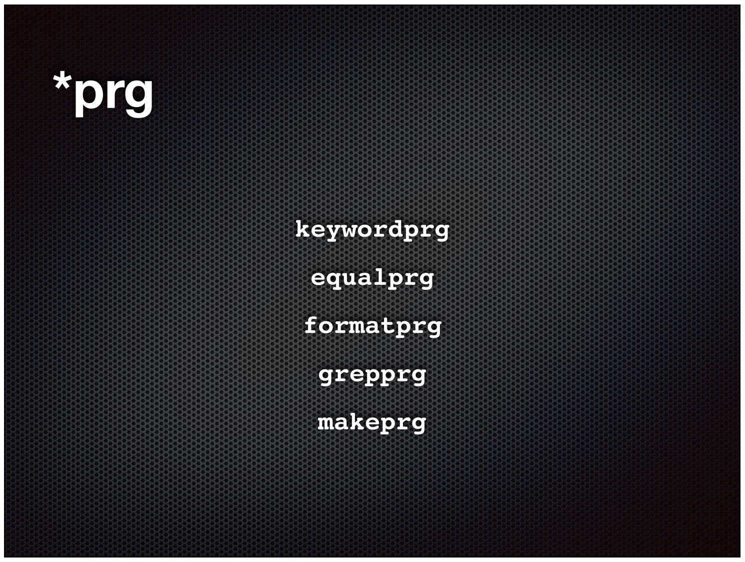
```
:r!cut -d : -f 1,7 /etc/passwd
```


Think Inside the Box

- grep
- sort
- rot13
- par / fmt
- hostname
- date / time
- uniq
- expand / unexpand
- ...and many more

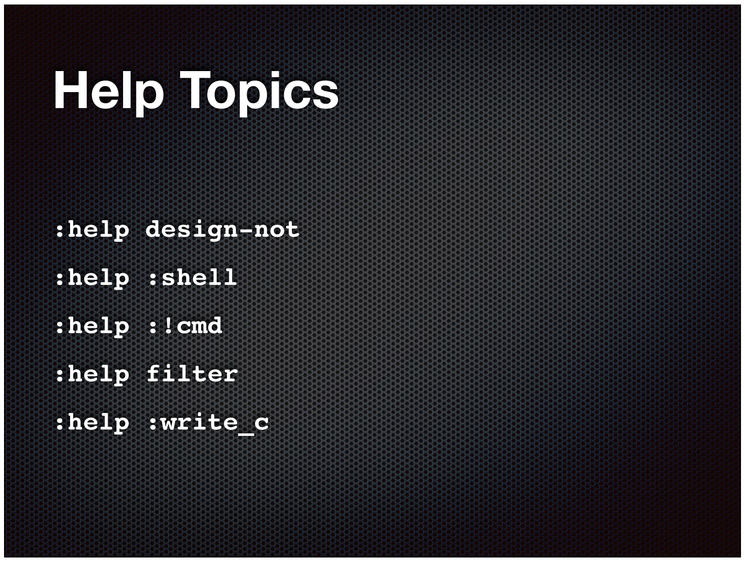
All of these commands have equivalents built in to Vim.

Vim includes a lot more functionality than the original vi, and therefore doesn't have to depend on as many external commands as it once did. While there's nothing wrong with using external commands instead of the built-in capabilities, it's often faster and simpler to use the stuff that's baked into Vim. Take a look at `:h functions` to get an idea of what's available.


*prg

```
keywordprg
equalprg
formatprg
grepprg
makeprg
```

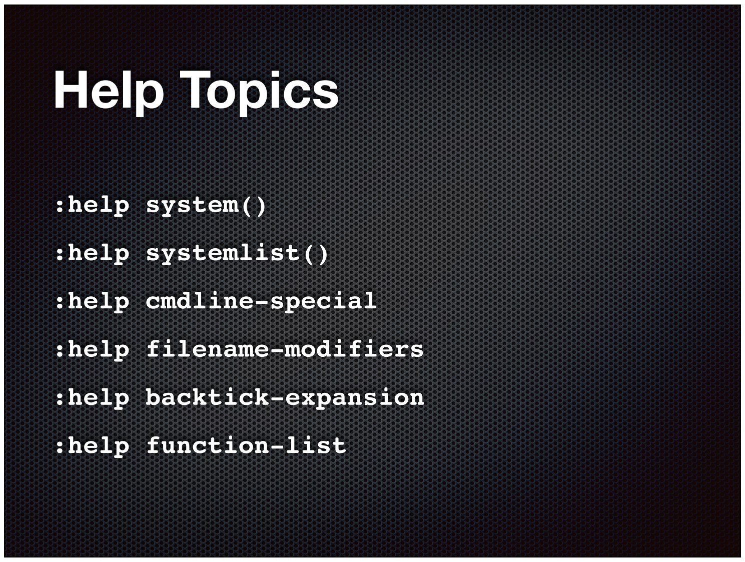
Vim can run commands for you behind the scenes, often as a fallback. These are some of the settings that control the commands that Vim will run.



Help Topics

```
:help design-not
:help :shell
:help :!cmd
:help filter
:help :write_c
```

As always, the Vim help is very comprehensive. These are just a few of the relevant topics.



Help Topics

```
:help system()
:help systemlist()
:help cmdline-special
:help filename-modifiers
:help backtick-expansion
:help function-list
```

As always, the Vim help is very comprehensive. These are just a few of the relevant topics.

?

Any questions or comments?

Thanks!