

Greybeard's Guide to *NIX

or, how I learned to love the command line

Steve Roggenkamp

Institute for Biomedical Informatics
University of Kentucky

`steve.roggenkamp@uky.edu`

`https://github.com/roggenkamps/GreybeardsGuide`

13 Sep 2017



Why???

The command line is Ancient and Decrepit? Why waste time on it?

- Systems administration
- Resource constrained systems
 - Embedded systems
 - High performance systems
- It can do a lot for you

Why???

The command line is Ancient and Decrepit? Why waste time on it?

- Systems administration
- Resource constrained systems
 - Embedded systems
 - High performance systems
- It can do a lot for you

Why???

The command line is Ancient and Decrepit? Why waste time on it?

- Systems administration
- Resource constrained systems
 - Embedded systems
 - High performance systems
- It can do a lot for you

Why???

The command line is Ancient and Decrepit? Why waste time on it?

- Systems administration
- Resource constrained systems
 - Embedded systems
 - High performance systems
- It can do a lot for you

Why???

The command line is Ancient and Decrepit? Why waste time on it?

- Systems administration
- Resource constrained systems
 - Embedded systems
 - High performance systems
- It can do a lot for you

What we're going to cover

- Six essential concepts to efficiently use command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Solve a practical problem with just a line or two of code

What we're going to cover

- Six essential concepts to efficiently use command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Solve a practical problem with just a line or two of code

What we're going to cover

- Six essential concepts to efficiently use command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Solve a practical problem with just a line or two of code

What we're going to cover

- Six essential concepts to efficiently use command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Solve a practical problem with just a line or two of code

What we're going to cover

- Six essential concepts to efficiently use command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Solve a practical problem with just a line or two of code

What we're going to cover

- Six essential concepts to efficiently use command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Solve a practical problem with just a line or two of code

What we're going to cover

- Six essential concepts to efficiently use command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Solve a practical problem with just a line or two of code

What we're going to cover

- Six essential concepts to efficiently use command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Solve a practical problem with just a line or two of code

The Shell, a must have

- Parses your input and executes commands, think REPL
- Provides an environment in which you work
- Provides a scripting programming language to automate tasks

The Shell, a must have

- Parses your input and executes commands, think REPL
- Provides an environment in which you work
- Provides a scripting programming language to automate tasks

The Shell, a must have

- Parses your input and executes commands, think REPL
- Provides an environment in which you work
- Provides a scripting programming language to automate tasks

Shells - If you don't like one, try another

- `ash`, `bash`, `dash`
- Bourne shell, Korn shell, PD Korn, Bourne-again shell
- `csh`, `tcsh`, `zsh`
- Busybox (which normally uses `ash`)
- Each provides unique set of capabilities
- Today's talk focuses on Bourne-again shell, aka, `bash`
 - Default shell for most Linux and Mac OSX
 - Available for most *NIX system

Shells - If you don't like one, try another

- `ash`, `bash`, `dash`
- Bourne shell, Korn shell, PD Korn, Bourne-again shell
- `csh`, `tcsh`, `zsh`
- Busybox (which normally uses `ash`)
- Each provides unique set of capabilities
- Today's talk focuses on Bourne-again shell, aka, `bash`
 - Default shell for most Linux and Mac OSX
 - Available for most *NIX system

Shells - If you don't like one, try another

- `ash`, `bash`, `dash`
- Bourne shell, Korn shell, PD Korn, Bourne-again shell
- `csh`, `tcsh`, `zsh`
- Busybox (which normally uses `ash`)
- Each provides unique set of capabilities
- Today's talk focuses on Bourne-again shell, aka, `bash`
 - Default shell for most Linux and Mac OSX
 - Available for most *NIX system

Shells - If you don't like one, try another

- `ash`, `bash`, `dash`
- Bourne shell, Korn shell, PD Korn, Bourne-again shell
- `csh`, `tcsh`, `zsh`
- Busybox (which normally uses `ash`)
- Each provides unique set of capabilities
- Today's talk focuses on Bourne-again shell, aka, `bash`
 - Default shell for most Linux and Mac OSX
 - Available for most *NIX system

Shells - If you don't like one, try another

- `ash`, `bash`, `dash`
- Bourne shell, Korn shell, PD Korn, Bourne-again shell
- `csh`, `tcsh`, `zsh`
- Busybox (which normally uses `ash`)
- Each provides unique set of capabilities
- Today's talk focuses on Bourne-again shell, aka, `bash`
 - Default shell for most Linux and Mac OSX
 - Available for most *NIX system

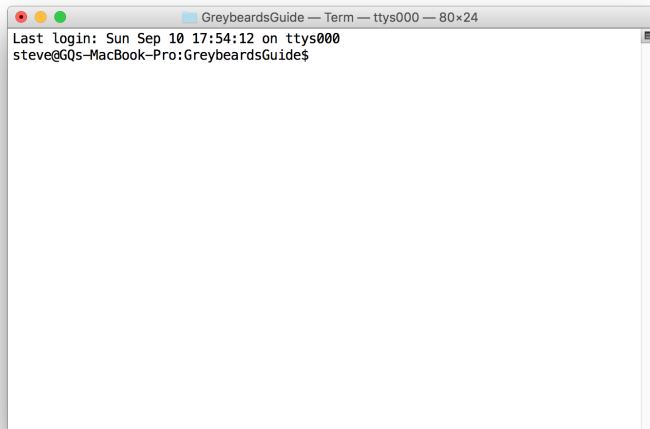
Shells - If you don't like one, try another

- `ash`, `bash`, `dash`
- Bourne shell, Korn shell, PD Korn, Bourne-again shell
- `csh`, `tcsh`, `zsh`
- Busybox (which normally uses `ash`)
- Each provides unique set of capabilities
- Today's talk focuses on Bourne-again shell, aka, `bash`
 - Default shell for most Linux and Mac OSX
 - Available for most *NIX system

Starting a shell

- Start a terminal in a GUI environment
- Remotely execute a shell via `ssh`
- Log into a system console

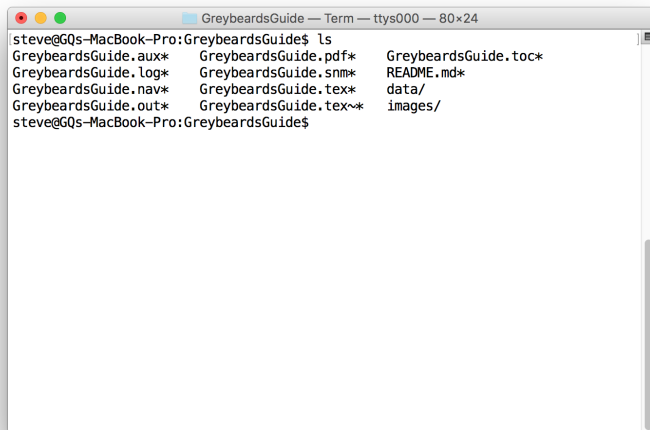
Sample session start

A screenshot of a macOS Terminal window. The title bar at the top reads "GreybeardsGuide — Term — ttys000 — 80x24". The terminal content shows the last login message: "Last login: Sun Sep 10 17:54:12 on ttys000". Below this, the prompt "steve@GQs-MacBook-Pro:GreybeardsGuide\$" is displayed, with the cursor at the end of the line.

```
GreybeardsGuide — Term — ttys000 — 80x24
Last login: Sun Sep 10 17:54:12 on ttys000
steve@GQs-MacBook-Pro:GreybeardsGuide$
```

Now what?? Coder's Block!!!

Session - list current directory



A terminal window titled "GreybeardsGuide — Term — ttys000 — 80x24" is shown. The prompt is "steve@GQs-MacBook-Pro:GreybeardsGuide\$". The command "ls" has been entered, and the output is displayed in three columns:

```
steve@GQs-MacBook-Pro:GreybeardsGuide$ ls
GreybeardsGuide.aux*  GreybeardsGuide.pdf*  GreybeardsGuide.toc*
GreybeardsGuide.log*  GreybeardsGuide.snm*  README.md*
GreybeardsGuide.nav*  GreybeardsGuide.tex*  data/
GreybeardsGuide.out*  GreybeardsGuide.tex~*  images/
steve@GQs-MacBook-Pro:GreybeardsGuide$
```

`ls` lists the contents of a directory

Shell - file redirection



```
data — skro232@skro232: ~ — ttys003 — 80x12
steve$ echo Hello world! >hello.txt
steve$ echo "A second line" >> hello.txt
steve$ cat hello.txt
Hello world!
A second line
steve$ cat < hello.txt
Hello world!
A second line
steve$
```

- > redirects a program's standard output to a file
- >> appends a program's standard output to a file
- < redirects a file to a program's standard input

Shell - executing scripts

```
data — skro232@skro232: ~ — ttys003 — 80x26
[steve$ echo 'ls -l' > cmd1.txt
[steve$ . cmd1.txt
total 16
-rw-r--r--  1 steve  staff   6 Sep 13 13:51 cmd1.txt
-rw-r--r--  1 steve  staff  27 Sep 13 08:28 hello.txt
[steve$ cat cmd1.txt
ls -l
[steve$ echo '#!/bin/bash' > cmd2.txt
[steve$ echo 'ls -l' >> cmd2.txt
[steve$ cat cmd2.txt
#!/bin/bash
ls -l
[steve$ . cmd2.txt
total 24
-rw-r--r--  1 steve  staff   6 Sep 13 13:51 cmd1.txt
-rw-r--r--  1 steve  staff  18 Sep 13 13:52 cmd2.txt
-rw-r--r--  1 steve  staff  27 Sep 13 08:28 hello.txt
[steve$ ./cmd2.txt
-bash: ./cmd2.txt: Permission denied
[steve$ chmod +x cmd2.txt
[steve$ ./cmd2.txt
total 24
-rw-r--r--  1 steve  staff   6 Sep 13 13:51 cmd1.txt
-rwxr-xr-x  1 steve  staff  18 Sep 13 13:52 cmd2.txt
-rw-r--r--  1 steve  staff  27 Sep 13 08:28 hello.txt
steve$ █
```

Shell - filename patterns



```
data — skro232@skro232: ~ — ttys003 — 80x12
[steve$ ls
cmd.txt          cmd2.txt          hello.txt
cmd1.txt         cmd3.sh           regexp
[steve$ ls cmd.txt
cmd.txt
[steve$ ls cmd?.txt
cmd1.txt         cmd2.txt
[steve$ ls cmd*
cmd.txt          cmd1.txt          cmd2.txt          cmd3.sh
[steve$ ls cmd*.txt
cmd.txt          cmd1.txt          cmd2.txt
steve$
```

- *NIX does not respect “filename extensions”
- 'cmd.txt' matches a file named 'cmd.txt'
- ? matches a single character
- * matches any numbers of characters

Shell - filename patterns, cont.

A terminal window titled 'data — skro232@skro232: ~ — ttys003 — 80x12' showing a series of 'ls' commands and their outputs. The commands use various shell filename patterns to filter files. The outputs show files like 'cmd1.txt', 'cmd2.txt', and 'cmd3.sh' being listed or filtered out.

```
steve$ ls cmd[1-9]*
cmd1.txt      cmd2.txt      cmd3.sh
steve$ ls cmd[1-2]*
cmd1.txt      cmd2.txt
steve$ ls cmd[12]*
cmd1.txt      cmd2.txt
steve$ ls cmd[13]*
cmd1.txt      cmd3.sh
steve$ ls cmd[13]????
cmd1.txt
steve$
```

- `[xy]` matches characters 'x' and 'y'
- `[x-z]` matches characters between 'x' and 'z' inclusive

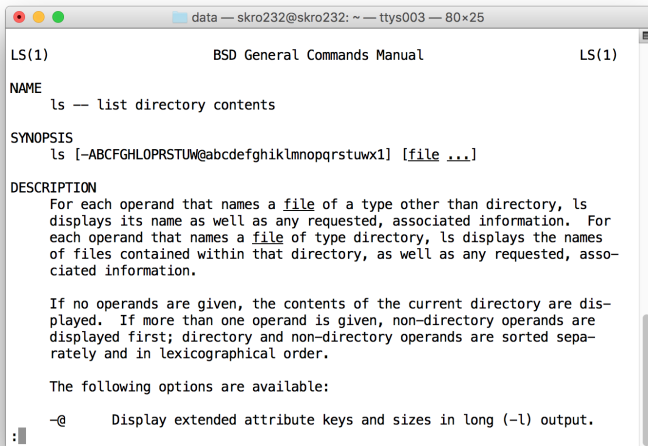
Pipes - Composing programs

- Pipes feed the output from a program to the input of another program
- Allow us to string programs together for “one-of” programs
- Very efficient
 - Creates one process per program
 - File buffer between programs
 - Signals and scheduler coordinate writing and reading data between programs - no extra files needed

Pipe examples

```
data — skro232@skro232: ~ — ttys003 — 80x24
steve$ ls -l /usr/bin | wc -l
1055
steve$ ls -l /usr/bin | sort -nr -k 5 | head -5
-r-xr-xr-x 1 root wheel 11951776 Jun 23 07:13 emacs*
-rwxr-xr-x 1 root wheel 10606096 Jun 23 07:13 php*
-r-xr-xr-x 1 root wheel 5637248 May 5 2016 perl5.18*
-r-xr-xr-x 1 root wheel 5477088 May 5 2016 perl5.16*
-rwxr-xr-x 1 root wheel 3201184 Jul 8 2016 emacs-undumped*
steve$ ls -l /usr/bin | sort -nr -k 5 | grep May | head -5
-r-xr-xr-x 1 root wheel 5637248 May 5 2016 perl5.18*
-r-xr-xr-x 1 root wheel 5477088 May 5 2016 perl5.16*
-r-xr-xr-x 1 root wheel 2242256 May 5 2016 db_printlog*
-r-xr-xr-x 1 root wheel 2195504 May 5 2016 db_codegen*
-r-xr-xr-x 1 root wheel 2192016 May 5 2016 db_load*
steve$ ls -l /usr/bin | sort -nr -k 5 | grep 2015 | head -5
-rwxr-xr-x 1 root wheel 162321 Aug 1 2015 afmtodit*
-r-xr-xr-x 1 root wheel 82203 Aug 2 2015 tkpp5.18*
-r-xr-xr-x 1 root wheel 80592 Aug 6 2015 gen_bridge_metadata*
-r-xr-xr-x 1 root wheel 78220 Aug 2 2015 tkpp5.16*
-rwxr-xr-x 1 root wheel 60725 Aug 11 2015 h2xs5.18*
steve$
```


What to type for a given command



```
data — skro232@skro232: ~ — ttys003 — 80x25
LS(1)                                BSD General Commands Manual                                LS(1)

NAME
  ls — list directory contents

SYNOPSIS
  ls [-ABCFGHLOPRSTUW@abcdefghijklmnopqrstuvwxyz1] [file ...]

DESCRIPTION
  For each operand that names a file of a type other than directory, ls
  displays its name as well as any requested, associated information. For
  each operand that names a file of type directory, ls displays the names
  of files contained within that directory, as well as any requested, asso-
  ciated information.

  If no operands are given, the contents of the current directory are dis-
  played. If more than one operand is given, non-directory operands are
  displayed first; directory and non-directory operands are sorted sepa-
  rately and in lexicographical order.

  The following options are available:

  -@      Display extended attribute keys and sizes in long (-l) output.
```

• `man command` provides the manual page for *command*

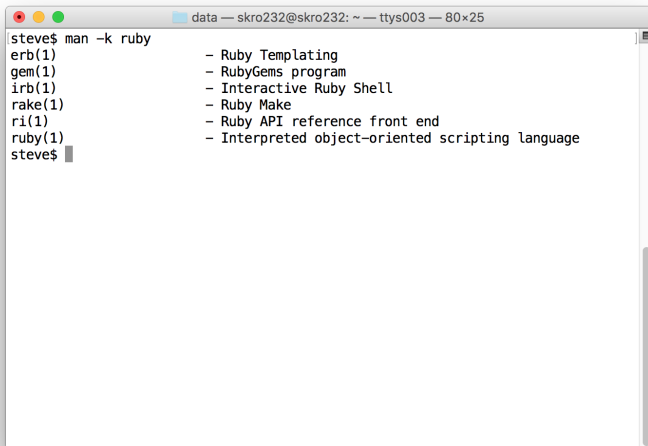
Man pages

- `man` pages contain multiple sections
- Typical sections include:
 - NAME - name of the command and brief discussion
 - SYNOPSIS - how to invoke the command and list of arguments
 - DESCRIPTION - detailed description of the program
 - OPTIONS - options and their description
 - EXAMPLES - example invocations and what they do
 - EXIT STATUS - `exit(2)` codes (0 indicates success)
 - ENVIRONMENT - how shell variables affect program
 - SEE ALSO - other programs or documents to consult
 - BUGS - known issues

Man pages - cont.

- UNIX documentation system
- Organized into multiple sections
 - 1 - programs or shell commands
 - 2 - System (kernel) calls
 - 3 - Library calls
 - 4 - Special files (usually found in /dev)
 - 5 - File formats
 - 6 - Games
 - 7 - Miscellaneous
 - 8 - System administration commands (root)
 - 9 - Non-standard kernel commands

Man pages - How to find commands with 'man -k'



A terminal window titled 'data — skro232@skro232: ~ — ttys003 — 80x25' displays the command 'man -k ruby' and its output. The output lists several Ruby-related commands with their descriptions:

```
steve$ man -k ruby
erb(1)          - Ruby Templating
gem(1)          - RubyGems program
irb(1)          - Interactive Ruby Shell
rake(1)         - Ruby Make
ri(1)           - Ruby API reference front end
ruby(1)         - Interpreted object-oriented scripting language
steve$
```

Regular expressions - searching text

- Regular expressions allow you to search using string patterns
- Many programs in *NIX use regular expressions
- Most characters match themselves, 'abc' matches the string "abc"
- Meta characters:
 - ^ matches the beginning of a line
 - \$ matches the end of a line
 - . matches a single character
 - ? matches zero or one of the preceding pattern
 - * matches zero or more of the preceding pattern
 - + matches zero or more of the preceding pattern
 - [and] indicate the start and end of a character class
 - (and) indicate the start and end of an atom
- \ escapes meta characters: \\$ matches \$

Regular expressions - examples

- `/abc/` matches “abc”
- `/[a-z]/` matches a lower case character
- `/[a-z][A-Za-z0-9_]+/` matches identifiers in many languages
- `/^[0-9]+$` matches a line containing only a single integer
- `/^a.*z$/` matches a line starting with 'a' and ending with 'z'

Regular expressions - grep

- `grep` (Global Regular Expression Print) is the standard search program
- `grep -i PATTERN FILES` - ignores case
- `grep -l PATTERN FILES` - prints just the FILES containing PATTERN
- `grep -f RE_FILE FILES` - reads regular expressions, one per line, from RE_FILE

find - Query your file system

- `find` traverses a file system performing an action on each file it matches
- It provides many filters to fine tune a query:
 - file name pattern
 - file types (regular file, directories, links)
 - file properties (create time, modification time, etc.)
 - owner, group
 - number of links
 - and more
- Default action is print the name of the file

find - Example

```
ibisite — skro232@skro232: ~ — ttys003 — 80x24
steve$ find . -name '*.rb' | xargs grep -li menu
./app/controllers/menus_controller.rb
./app/helpers/menus_helper.rb
./app/models/menu.rb
./app/views/layouts/_header.html.erb
./app/views/menus/_form.html.erb
./app/views/menus/_new_menu.html.erb
./app/views/menus/_new_sub_menu.html.erb
./app/views/menus/add_menu.js.erb
./app/views/menus/add_sub_menu.js.erb
./app/views/menus/destroy.js.erb
./app/views/menus/edit.html.erb
./app/views/menus/index.html.erb
./app/views/menus/new.html.erb
./app/views/menus/show.html.erb
./config/initializers/authorization.rb
./config/routes.rb
./db/migrate/20160825183810_create_menus.rb
./db/schema.rb
./db/seeds_prev.rb
steve$
```

find - Example using sed to filter



```
ibisite — skro232@skro232: ~ — ttys003 — 80x25
steve$ find . -name '*.rb' | sed -e '/app.views/d' | xargs grep -li menu
./app/controllers/menus_controller.rb
./app/helpers/menus_helper.rb
./app/models/menu.rb
./config/initializers/authorization.rb
./config/routes.rb
./db/migrate/20160825183810_create_menus.rb
./db/schema.rb
./db/seeds_prev.rb
steve$
```

Command substitution - use the shell to write your commands

- `$ (pipe)` executes *pipe* and substitutes the text in place

Command substitution - cont

```
ibisite — skro232@skro232: ~ — ttys003 — 80x25
steve$ echo ls -l $(find . -name '*.rb' | sed -e '/app.views/d' | xargs grep -li menu)
ls -l ./app/controllers/menus_controller.rb ./app/helpers/menus_helper.rb ./app/
models/menu.rb ./config/initializers/authorization.rb ./config/routes.rb ./db/mi
grate/20160825183810_create_menus.rb ./db/schema.rb ./db/seeds_prev.rb
steve$
steve$
steve$ ls -l $(find . -name '*.rb' | sed -e '/app.views/d' | xargs grep -li menu)

-rw-r--r--  1 steve  staff   2481 Sep 16  2016 ./app/controllers/menus_controll
r.rb
-rw-r--r--  1 steve  staff    623 Sep  6  2016 ./app/helpers/menus_helper.rb
-rw-r--r--  1 steve  staff    120 Sep  6  2016 ./app/models/menu.rb
-rw-r--r--  1 steve  staff    853 Aug 29 14:50 ./config/initializers/authorizati
on.rb
-rw-r--r--  1 steve  staff   2143 Sep 12 10:38 ./config/routes.rb
-rw-r--r--  1 steve  staff    246 Sep  6  2016 ./db/migrate/20160825183810_creat
e_menus.rb
-rw-r--r--  1 steve  staff  11751 Sep  6 13:18 ./db/schema.rb
-rw-r--r--  1 steve  staff   2391 Sep 12 16:57 ./db/seeds_prev.rb
steve$
```

Command substitution - cont

```
ibisite — skro232@skro232: ~ — ttys003 — 80x25
steve$ ls -l $(find . -name '*.rb' | sed -e '/app.views/d' | xargs grep -li menu) |
| sort -nr -k 5 | head -4
-rw-r--r--  1 steve  staff   11751 Sep  6 13:18 ./db/schema.rb
-rw-r--r--  1 steve  staff    2481 Sep 16 2016 ./app/controllers/menus_controlle
r.rb
-rw-r--r--  1 steve  staff    2391 Sep 12 16:57 ./db/seeds_prev.rb
-rw-r--r--  1 steve  staff    2143 Sep 12 10:38 ./config/routes.rb
steve$
```

Common programs

- `awk` - The AWK text processing language
- `file` - describe the type of a file
- `grep` - find strings in files
- `less` - display a file a screen at a time
- `ls` - list files
- `sed` - Stream editor
- `sort` - Sort lines of a file
- `uniq` - Filters out repeated lines in sorted files
- `wc` - Counts characters, words and lines of a file
- `xargs` - used with `find` to execute a command on multiple files

Common programs

- `awk` - The AWK text processing language
- `file` - describe the type of a file
- `grep` - find strings in files
- `less` - display a file a screen at a time
- `ls` - list files
- `sed` - Stream editor
- `sort` - Sort lines of a file
- `uniq` - Filters out repeated lines in sorted files
- `wc` - Counts characters, words and lines of a file
- `xargs` - used with `find` to execute a command on multiple files

Common programs

- `awk` - The AWK text processing language
- `file` - describe the type of a file
- `grep` - find strings in files
- `less` - display a file a screen at a time
- `ls` - list files
- `sed` - Stream editor
- `sort` - Sort lines of a file
- `uniq` - Filters out repeated lines in sorted files
- `wc` - Counts characters, words and lines of a file
- `xargs` - used with `find` to execute a command on multiple files

Common programs

- `awk` - The AWK text processing language
- `file` - describe the type of a file
- `grep` - find strings in files
- `less` - display a file a screen at a time
- `ls` - list files
- `sed` - Stream editor
- `sort` - Sort lines of a file
- `uniq` - Filters out repeated lines in sorted files
- `wc` - Counts characters, words and lines of a file
- `xargs` - used with `find` to execute a command on multiple files

Common programs

- `awk` - The AWK text processing language
- `file` - describe the type of a file
- `grep` - find strings in files
- `less` - display a file a screen at a time
- `ls` - list files
- `sed` - Stream editor
- `sort` - Sort lines of a file
- `uniq` - Filters out repeated lines in sorted files
- `wc` - Counts characters, words and lines of a file
- `xargs` - used with `find` to execute a command on multiple files

Common programs

- `awk` - The AWK text processing language
- `file` - describe the type of a file
- `grep` - find strings in files
- `less` - display a file a screen at a time
- `ls` - list files
- `sed` - Stream editor
- `sort` - Sort lines of a file
- `uniq` - Filters out repeated lines in sorted files
- `wc` - Counts characters, words and lines of a file
- `xargs` - used with `find` to execute a command on multiple files

Common programs

- `awk` - The AWK text processing language
- `file` - describe the type of a file
- `grep` - find strings in files
- `less` - display a file a screen at a time
- `ls` - list files
- `sed` - Stream editor
- `sort` - Sort lines of a file
- `uniq` - Filters out repeated lines in sorted files
- `wc` - Counts characters, words and lines of a file
- `xargs` - used with `find` to execute a command on multiple files

Common programs

- `awk` - The AWK text processing language
- `file` - describe the type of a file
- `grep` - find strings in files
- `less` - display a file a screen at a time
- `ls` - list files
- `sed` - Stream editor
- `sort` - Sort lines of a file
- `uniq` - Filters out repeated lines in sorted files
- `wc` - Counts characters, words and lines of a file
- `xargs` - used with `find` to execute a command on multiple files

Common programs

- `awk` - The AWK text processing language
- `file` - describe the type of a file
- `grep` - find strings in files
- `less` - display a file a screen at a time
- `ls` - list files
- `sed` - Stream editor
- `sort` - Sort lines of a file
- `uniq` - Filters out repeated lines in sorted files
- `wc` - Counts characters, words and lines of a file
- `xargs` - used with `find` to execute a command on multiple files

Common programs

- `awk` - The AWK text processing language
- `file` - describe the type of a file
- `grep` - find strings in files
- `less` - display a file a screen at a time
- `ls` - list files
- `sed` - Stream editor
- `sort` - Sort lines of a file
- `uniq` - Filters out repeated lines in sorted files
- `wc` - Counts characters, words and lines of a file
- `xargs` - used with `find` to execute a command on multiple files

Example problem - Solve NPR Sunday puzzle

From NPR Weekend Edition Sunday, Sept 4, 2016.

Next week's challenge, from listener Norm Baird of Toledo, Wash.: If you squish the small letters "r" and "n" too closely together, they look like an "m." Think of a common five-letter word with the consecutive letters "r" and "n" that becomes its own opposite if you change them to an "m."

About 840 people solved this puzzle.

Let's use our Ancient and Decrepit Tools to solve it in just two lines of code.


```
data — skro232@skro232: ~ — ttys003 — 80x25
steve$ egrep '^.....$' /usr/share/dict/words | wc -l
10230
steve$ egrep '^.....$' /usr/share/dict/words | sed -n -e 's/\(.*\)rn\(.*\)\/^1m\2$/p' > regexp
steve$ head -2 regexp
^acom$
^adom$
steve$ grep -f regexp /usr/share/dict/words | sed -e 's/\(.*\)m\(.*\)\/1rn\2 -> \1m\2/'
churn -> chum
herne -> heme
horny -> homy
learn -> leam
morne -> mome
scarn -> scam
sharn -> sham
starn -> stam
stern -> stem
sworn -> swom
steve$ wc -l regexp
59 regexp
steve$ █
```

Answer: STERN and STEM

Wrap up

- Covered why you might want to use the command line
- Described six concepts essential to efficient use of command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Demonstrated efficient command line use to solve an NPR Sunday puzzle in two lines of code

Wrap up

- Covered why you might want to use the command line
- Described six concepts essential to efficient use of command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Demonstrated efficient command line use to solve an NPR Sunday puzzle in two lines of code

Wrap up

- Covered why you might want to use the command line
- Described six concepts essential to efficient use of command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Demonstrated efficient command line use to solve an NPR Sunday puzzle in two lines of code

Wrap up

- Covered why you might want to use the command line
- Described six concepts essential to efficient use of command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Demonstrated efficient command line use to solve an NPR Sunday puzzle in two lines of code

Wrap up

- Covered why you might want to use the command line
- Described six concepts essential to efficient use of command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Demonstrated efficient command line use to solve an NPR Sunday puzzle in two lines of code

Wrap up

- Covered why you might want to use the command line
- Described six concepts essential to efficient use of command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Demonstrated efficient command line use to solve an NPR Sunday puzzle in two lines of code

Wrap up

- Covered why you might want to use the command line
- Described six concepts essential to efficient use of command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Demonstrated efficient command line use to solve an NPR Sunday puzzle in two lines of code

Wrap up

- Covered why you might want to use the command line
- Described six concepts essential to efficient use of command line:
 - Command shell
 - Pipes to connect commands
 - Man pages for documentation
 - Regular expressions for text processing
 - `find` to query file systems
 - Command substitution
- Demonstrated efficient command line use to solve an NPR Sunday puzzle in two lines of code

Questions?