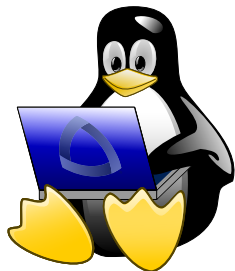# Readline Ninja Skills

Jack Rosenthal

7 March 2016

**Colorado School of Mines**
Linux Users Group

# Readline

- A library for interactive line editing that your shell probably uses.

- Responsible for things like tab completion, history expansion, and all of those useful keystrokes

- Readline saves you keystrokes.

- Some readline things can make you look like a total ninja.

- Some readline things make you feel like a total ninja.

# Readline

- A library for interactive line editing that your shell probably uses.
- Responsible for things like tab completion, history expansion, and all of those useful keystrokes
- Readline saves you keystrokes.
- Some readline things can make you look like a total ninja.
- Some readline things make you feel like a total ninja.

# Readline

- A library for interactive line editing that your shell probably uses.
- Responsible for things like tab completion, history expansion, and all of those useful keystrokes
- Readline saves you keystrokes.
- Some readline things can make you look like a total ninja.
- Some readline things make you feel like a total ninja.

# Readline

- A library for interactive line editing that your shell probably uses.
- Responsible for things like tab completion, history expansion, and all of those useful keystrokes
- Readline saves you keystrokes.
- Some readline things can make you look like a total ninja.
- Some readline things make you feel like a total ninja.

# Readline

- A library for interactive line editing that your shell probably uses.
- Responsible for things like tab completion, history expansion, and all of those useful keystrokes
- Readline saves you keystrokes.
- Some readline things can make you look like a total ninja.
- Some readline things make you feel like a total ninja.

# History

Readline can track your history, most shells let you use the `history` builtin to view your history.
You can navigate your history using the up and down keys.

# Tab completion

Most of us already know what this and would die without it.

# Event Designators

- ! - begin history expansion
- !! - refer to the last command
- !*n* - refer to the *n*-th command in history
- !−*n* - refer to the current command minus *n*
- !# - refer to the current command you are typing
- !*search* - refer to the last commmand that starts with *search*
  !?*search*? - refer to the last command with *search* anywhere in the command

Examples:

# Event Designators

- ! - begin history expansion
- !! - refer to the last command
- !*n* - refer to the *n*-th command in history
- !−*n* - refer to the current command minus *n*
- !# - refer to the current command you are typing
- !*search* - refer to the last commmand that starts with *search*
  !?*search*? - refer to the last command with *search* anywhere in the command

Examples:

# Event Designators

- ! - begin history expansion
- !! - refer to the last command
- !$n$ - refer to the $n$-th command in history
- !$-n$ - refer to the current command minus $n$
- !# - refer to the current command you are typing
- !$search$ - refer to the last commmand that starts with $search$
  !?$search$? - refer to the last command with $search$ anywhere in the command

Examples:

# Event Designators

- ! - begin history expansion
- !! - refer to the last command
- !*n* - refer to the *n*-th command in history
- !−*n* - refer to the current command minus *n*
- !# - refer to the current command you are typing
- !*search* - refer to the last commmand that starts with *search*
  !?*search*? - refer to the last command with *search* anywhere
  in the command

Examples:

# Event Designators

- `!` - begin history expansion
- `!!` - refer to the last command
- `!`*n* - refer to the *n*-th command in history
- `!-`*n* - refer to the current command minus *n*
- `!#` - refer to the current command you are typing
- `!`*search* - refer to the last commmand that starts with *search*
  `!?`*search*`?` - refer to the last command with *search* anywhere in the command

Examples:

# Event Designators

- ! - begin history expansion
- !! - refer to the last command
- !*n* - refer to the *n*-th command in history
- !-*n* - refer to the current command minus *n*
- !# - refer to the current command you are typing
- !*search* - refer to the last commmand that starts with *search*

  !?*search*? - refer to the last command with *search* anywhere in the command

Examples:

# Event Designators

- `!` - begin history expansion
- `!!` - refer to the last command
- `!`*n* - refer to the *n*-th command in history
- `!-`*n* - refer to the current command minus *n*
- `!#` - refer to the current command you are typing
- `!`*search* - refer to the last commmand that starts with *search*

  `!?`*search*`?` - refer to the last command with *search* anywhere in the command

Examples:

# Event Designators

- ! - begin history expansion
- !! - refer to the last command
- !*n* - refer to the *n*-th command in history
- !−*n* - refer to the current command minus *n*
- !# - refer to the current command you are typing
- !*search* - refer to the last commmand that starts with *search*

  !?*search*? - refer to the last command with *search* anywhere in the command

Examples:

# Event Designators

- `!` - begin history expansion
- `!!` - refer to the last command
- `!`*n* - refer to the *n*-th command in history
- `!-`*n* - refer to the current command minus *n*
- `!#` - refer to the current command you are typing
- `!`*search* - refer to the last commmand that starts with *search*

  `!?`*search*`?` - refer to the last command with *search* anywhere in the command

Examples:

- `sudo !!` - run the last command with `sudo` in front
- `!grep` - run the last command you typed beginning with `grep`

# Event Designators

- `!` - begin history expansion
- `!!` - refer to the last command
- `!`*n* - refer to the *n*-th command in history
- `!-`*n* - refer to the current command minus *n*
- `!#` - refer to the current command you are typing
- `!`*search* - refer to the last commmand that starts with *search*

  `!?`*search*`?` - refer to the last command with *search* anywhere in the command

Examples:

- `sudo !!` - run the last command with `sudo` in front
- `!grep` - run the last command you typed beginning with `grep`

Often times you will want only part of a command, so you can use word designators to select which parts you want. Follow an event designator with a colon (:) and then a word designator.

- :n - select argument n (zero indexed)
- n* - select everything starting at n
- :$ - select the last argument (choice of a suggest...)
- :* - select all arguments, starting from one (synonym for 1*)
- :-n - select the arguments from zero inclusive to n

Examples:

- !!:$ - select the last argument of the last command
- !echo:1* - take the 1st to the last argument of your latest echo use

# Word Designators

Often times you will want only part of a command, so you can use word designators to select which parts you want. Follow an event designator with a colon (:) and then a word designator.

# Word Designators

Often times you will want only part of a command, so you can use word designators to select which parts you want. Follow an event designator with a colon (:) and then a word designator.

- :*n* - select argument *n* (zero indexed)
- :*n*–*m* - select arguments *n* through *m*
- :$ - select the last argument (think of a regex)
- :* - select all arguments, omitting the command name (equivalent to :1–$)
  :% - select the argument that matches ?*search*?

Examples:

# Word Designators

Often times you will want only part of a command, so you can use word designators to select which parts you want. Follow an event designator with a colon (:) and then a word designator.

- :*n* - select argument *n* (zero indexed)
- :*n*–*m* - select arguments *n* through *m*
- :$ - select the last argument (think of a regex)
- :* - select all arguments, omitting the command name (equivalent to :1–$)

:% - select the argument that matches ?*search*?

Examples:

# Word Designators

Often times you will want only part of a command, so you can use word designators to select which parts you want. Follow an event designator with a colon (:) and then a word designator.

- :*n* - select argument *n* (zero indexed)
- :*n*–*m* - select arguments *n* through *m*
- :$ - select the last argument (think of a regex)
- :* - select all arguments, omitting the command name (equivalent to :1–$)

  :% - select the argument that matches ?*search*?

Examples:

# Word Designators

Often times you will want only part of a command, so you can use word designators to select which parts you want. Follow an event designator with a colon (:) and then a word designator.

- :*n* - select argument *n* (zero indexed)
- :*n*–*m* - select arguments *n* through *m*
- :$ - select the last argument (think of a regex)
- :* - select all arguments, omitting the command name (equivalent to :1-$)

:% - select the argument that matches ?*search*?

Examples:

# Word Designators

Often times you will want only part of a command, so you can use word designators to select which parts you want. Follow an event designator with a colon (:) and then a word designator.

- :*n* - select argument *n* (zero indexed)
- :*n*–*m* - select arguments *n* through *m*
- :$ - select the last argument (think of a regex)
- :* - select all arguments, omitting the command name (equivalent to :1-$)

  :% - select the argument that matches ?*search*?

Examples:

# Word Designators

Often times you will want only part of a command, so you can use word designators to select which parts you want. Follow an event designator with a colon (:) and then a word designator.

- :*n* - select argument *n* (zero indexed)
- :*n*–*m* - select arguments *n* through *m*
- :$ - select the last argument (think of a regex)
- :* - select all arguments, omitting the command name (equivalent to :1-$)

  :% - select the argument that matches ?*search*?

Examples:

- cd !!:1 - cd to the first argument of the last command
- vim !-2:$ - edit the file that is the last argument of two commands ago

# Word Designators

Often times you will want only part of a command, so you can use word designators to select which parts you want. Follow an event designator with a colon (:) and then a word designator.

- :*n* - select argument *n* (zero indexed)
- :*n*–*m* - select arguments *n* through *m*
- :$ - select the last argument (think of a regex)
- :* - select all arguments, omitting the command name (equivalent to :1-$)

  :% - select the argument that matches ?*search*?

Examples:

- cd !!:1 - cd to the first argument of the last command.
- vim !-2:$ - edit the file that is the last argument of two commands ago

# Word Designators

Often times you will want only part of a command, so you can use word designators to select which parts you want. Follow an event designator with a colon (:) and then a word designator.

- :*n* - select argument *n* (zero indexed)
- :*n*–*m* - select arguments *n* through *m*
- :$ - select the last argument (think of a regex)
- :* - select all arguments, omitting the command name (equivalent to :1-$)

  :% - select the argument that matches ?*search*?

Examples:

- cd !!:1 - cd to the first argument of the last command.
- vim !-2:$ - edit the file that is the last argument of two commands ago

## Modifiers

Modifiers let you chop up the history expansion in ways that you like. You can chain any amount of modifiers that you would like onto your expansion.

- :r - Chop off the extension of a filename
- :h - Remove the filename component, leaving only the directory (think of head)
- :t - Remove the directory component, leaving only the filename (think of tail)
- :q - Quote each of the arguments
- :s/*search*/*replace*/ - sed style substitution
- :gs/*search*/*replace*/ - sed style substitution, globally
- :p - print the history expansion, don't execute quite yet

## Modifiers

Modifiers let you chop up the history expansion in ways that you like. You can chain any amount of modifiers that you would like onto your expansion.

- `:r` - Chop off the extension of a filename
- `:h` - Remove the filename component, leaving only the directory (think of head)
- `:t` - Remove the directory component, leaving only the filename (think of tail)
- `:q` - Quote each of the arguments
- `:s/search/replace/` - sed style substitution
- `:gs/search/replace/` - sed style substitution, globally
- `:p` - print the history expansion, don't execute quite yet

# Modifiers

Modifiers let you chop up the history expansion in ways that you like. You can chain any amount of modifiers that you would like onto your expansion.

- `:r` - Chop off the extension of a filename
- `:h` - Remove the filename component, leaving only the directory (think of head)
- `:t` - Remove the directory component, leaving only the filename (think of tail)
- `:q` - Quote each of the arguments
- `:s/`*search*`/`*replace*`/` - sed style substitution
- `:gs/`*search*`/`*replace*`/` - sed style substitution, globally
- `:p` - print the history expansion, don't execute quite yet

# Modifiers

Modifiers let you chop up the history expansion in ways that you like. You can chain any amount of modifiers that you would like onto your expansion.

- `:r` - Chop off the extension of a filename
- `:h` - Remove the filename component, leaving only the directory (think of head)
- `:t` - Remove the directory component, leaving only the filename (think of tail)
- `:q` - Quote each of the arguments
- `:s/`*search*`/`*replace*`/` - sed style substitution
- `:gs/`*search*`/`*replace*`/` - sed style substitution, globally
- `:p` - print the history expansion, don't execute quite yet

# Modifiers

Modifiers let you chop up the history expansion in ways that you like. You can chain any amount of modifiers that you would like onto your expansion.

- `:r` - Chop off the extension of a filename
- `:h` - Remove the filename component, leaving only the directory (think of head)
- `:t` - Remove the directory component, leaving only the filename (think of tail)
- `:q` - Quote each of the arguments
- `:s/`*search*`/`*replace*`/` - sed style substitution
- `:gs/`*search*`/`*replace*`/` - sed style substitution, globally
- `:p` - print the history expansion, don't execute quite yet

# Modifiers

Modifiers let you chop up the history expansion in ways that you
like. You can chain any amount of modifiers that you would like
onto your expansion.

- `:r` - Chop off the extension of a filename
- `:h` - Remove the filename component, leaving only the
  directory (think of head)
- `:t` - Remove the directory component, leaving only the
  filename (think of tail)
- `:q` - Quote each of the arguments
- `:s/search/replace/` - sed style substitution
- `:gs/search/replace/` - sed style substitution, globally
- `:p` - print the history expansion, don't execute quite yet

# Modifiers

Modifiers let you chop up the history expansion in ways that you
like. You can chain any amount of modifiers that you would like
onto your expansion.

- :r - Chop off the extension of a filename
- :h - Remove the filename component, leaving only the
  directory (think of head)
- :t - Remove the directory component, leaving only the
  filename (think of tail)
- :q - Quote each of the arguments
- :s/*search*/*replace*/ - sed style substitution
- :gs/*search*/*replace*/ - sed style substitution, globally
- :p - print the history expansion, don't execute quite yet

# Modifiers

Modifiers let you chop up the history expansion in ways that you like. You can chain any amount of modifiers that you would like onto your expansion.

- `:r` - Chop off the extension of a filename
- `:h` - Remove the filename component, leaving only the directory (think of head)
- `:t` - Remove the directory component, leaving only the filename (think of tail)
- `:q` - Quote each of the arguments
- `:s/`*search*`/`*replace*`/` - sed style substitution
- `:gs/`*search*`/`*replace*`/` - sed style substitution, globally
- `:p` - print the history expansion, don't execute quite yet

# Modifiers

- `:r` - Chop off the extension of a filename
- `:h` - Remove the filename component, leaving only the directory (think of head)
- `:t` - Remove the directory component, leaving only the filename (think of tail)
- `:q` - Quote each of the arguments
- `:s/`*search*`/`*replace*`/` - `sed` style substitution
- `:gs/`*search*`/`*replace*`/` - `sed` style substitution, globally
- `:p` - print the history expansion, don't execute quite yet

Examples:

- `mv important.png !#:1:r.gif` - rename important.png to important.gif
- `touch mydir/file.txt`
- `cd !$:h`

# Modifiers

- `:r` - Chop off the extension of a filename
- `:h` - Remove the filename component, leaving only the directory (think of head)
- `:t` - Remove the directory component, leaving only the filename (think of tail)
- `:q` - Quote each of the arguments
- `:s/`*search*`/`*replace*`/` - `sed` style substitution
- `:gs/`*search*`/`*replace*`/` - `sed` style substitution, globally
- `:p` - print the history expansion, don't execute quite yet

Examples:

- `mv important.png !#:1:r.gif` - rename important.png to important.gif
- `touch mydir/file.txt`
- `cd !$:h`

# Modifiers

- `:r` - Chop off the extension of a filename
- `:h` - Remove the filename component, leaving only the directory (think of head)
- `:t` - Remove the directory component, leaving only the filename (think of tail)
- `:q` - Quote each of the arguments
- `:s/`*search*`/`*replace*`/` - `sed` style substitution
- `:gs/`*search*`/`*replace*`/` - `sed` style substitution, globally
- `:p` - print the history expansion, don't execute quite yet

Examples:

- `mv important.png !#:1:r.gif` - rename `important.png` to `important.gif`
- `touch mydir/file.txt`
- `cd !$:h`

# Modifiers

- `:r` - Chop off the extension of a filename
- `:h` - Remove the filename component, leaving only the directory (think of head)
- `:t` - Remove the directory component, leaving only the filename (think of tail)
- `:q` - Quote each of the arguments
- `:s/`*search*`/`*replace*`/` - `sed` style substitution
- `:gs/`*search*`/`*replace*`/` - `sed` style substitution, globally
- `:p` - print the history expansion, don't execute quite yet

Examples:

- `mv important.png !#:1:r.gif` - rename `important.png` to `important.gif`
- `touch mydir/file.txt`
- `cd !$:h`

# Abbreviations Allowed

- !!:… can be shortened to !:…

- The : can be removed from word designators where it is unambiguous. So !$ and !* are allowed.

- The trailing / in a substitution can be omitted if it is unambigous that the substitution has ended.

- The trailing ? in a !?*search*? can be ommitted for the same reason.

- Any delimiter can be used in a substitution, so !:s*x*find*x*replace*x* is legal.

# Abbreviations Allowed

- `!!:…` can be shortened to `!:…`
- The `:` can be removed from word designators where it is unambiguous. So `!$` and `!*` are allowed.
- The trailing `/` in a substitution can be omitted if it is unambigous that the substitution has ended.
- The trailing `?` in a `!?`*search*`?` can be ommitted for the same reason.
- Any delimiter can be used in a substitution, so `!:s`x*find*x*replace*x is legal.

# Abbreviations Allowed

- !!:… can be shortened to !:…
- The : can be removed from word designators where it is unambiguous. So !$ and !* are allowed.
- The trailing / in a substitution can be omitted if it is unambigous that the substitution has ended.
- The trailing ? in a !?*search*? can be ommitted for the same reason.
- Any delimiter can be used in a substitution, so !:s*x*findx*replace*x is legal.

- `!!:`… can be shortened to `!:`…
- The `:` can be removed from word designators where it is unambiguous. So `!$` and `!*` are allowed.
- The trailing `/` in a substitution can be omitted if it is unambigous that the substitution has ended.
- The trailing `?` in a `!?`*search*`?` can be ommitted for the same reason.
- Any delimiter can be used in a substitution, so `!:s`x*find*x*replace*x is legal.

# Abbreviations Allowed

- `!!:`… can be shortened to `!:`…
- The `:` can be removed from word designators where it is unambiguous. So `!$` and `!*` are allowed.
- The trailing `/` in a substitution can be omitted if it is unambigous that the substitution has ended.
- The trailing `?` in a `!?`*search*`?` can be ommitted for the same reason.
- Any delimiter can be used in a substitution, so `!:s`x*find*x*replace*x is legal.

# Editing Modes

Readline provides editing modes similar to vi and emacs. Learn one and learn to love it. Most shells and programs have emacs as the default.

`<C-r>` (`vi: <Esc>/`) brings you to an search of your history.
`<C-s>` will reverse the direction of your search (You may need to
`stty -ixon`).

# C/C++ Readline Library

```c
#include <stdio.h>
#include <readline/readline.h>
#include <readline/history.h>

char * readline(const char *prompt);
```

Allocates memory to read a line, reads it from standard input (displaying `prompt` as the prompt line). Returns the line you read. You really should `free` the memory it allocated.

# Using History Features

```
void using_history(void);
```

Must be called before using history features.

```
int read_history(const char *filename);
int write_history(const char *filename);
```

For reading/writing saved history. Returns non-zero on failure and sets errno.

```
void add_history(const char *line);
```

Add a line to the history.

```
HIST_ENTRY ** histlst = history_list();
for (int i = 1; *histlst; i++, histlst++)
    printf("%d %s\n", i, (*histlst)->line);
```

List history.

## Using History Features

*void* using_history(*void*);

Must be called before using history features.

*int* read_history(const *char* *filename);
*int* write_history(const *char* *filename);

For reading/writing saved history. Returns non-zero on failure and sets errno.

*void* add_history(const *char* *line);

Add a line to the history.

```
HIST_ENTRY ** histlst = history_list();
for (int i = 1; *histlst; i++, histlst++)
    printf("%d %s\n", i, (*histlst)->line);
```

List history.

# Using History Features

*void* using_history(*void*);

Must be called before using history features.

*int* read_history(const *char* *filename);
*int* write_history(const *char* *filename);

For reading/writing saved history. Returns non-zero on failure and sets errno.

*void* add_history(const *char* *line);

Add a line to the history.

```
HIST_ENTRY ** histlst = history_list();
for (int i = 1; *histlst; i++, histlst++)
    printf("%d %s\n", i, (*histlst)->line);
```

List history.

## Using History Features

```
void using_history(void);
```

Must be called before using history features.

```
int read_history(const char *filename);
int write_history(const char *filename);
```

For reading/writing saved history. Returns non-zero on failure and
sets errno.

```
void add_history(const char *line);
```

Add a line to the history.

```
HIST_ENTRY ** histlst = history_list();
for (int i = 1; *histlst; i++, histlst++)
    printf("%d %s\n", i, (*histlst)->line);
```

List history.

# History Expansion (for free!)

`int history_expand(char *string, char **output);`

Expand string, placing the result into output, a pointer to a string.
Returns:

| | |
|---|---|
| 0 | If no expansions took place |
| 1 | If expansions did take place |
| -1 | If there was an error in expansion |
| 2 | If the line should be displayed, but not executed (:p) |

If an error occurred in expansion, then output contains a descriptive error message.

# A Complete Example

```
 1   #include <stdio.h>
 2   #include <stdlib.h>
 3   #include <unistd.h>
 4   #include <sys/wait.h>
 5   #include <readline/readline.h>
 6   #include <readline/history.h>
 7
 8   int main(void) {
 9       char *line = NULL, *expn = NULL;
10       int status;
11       using_history();
12       for (;;) {
13           free(line), free(expn);
14           line = readline("prompt> ");
15           if (!line) return 0; /* ^D to exit */
16           int expn_result = history_expand(line, &expn);
17           if (expn_result) puts(expn);
18           add_history(expn);
19           if (expn_result == 0 || expn_result == 1) {
20               int pid = fork();
21               if (pid < 0) return 1;
22               if (pid == 0) {
23                   char ** arg = history_tokenize(expn);
24                   execvp(*arg, arg);
25                   return 1;
26               }
27               waitpid(pid, &status, 0);
28           }
29       }
30       return 0;
31   }
```

# More Info

1. `man 3 readline`
2. man 3 history
3. RTFM
4. RTFM
5. RTFM
6. RTFM
7. RTFM

# More Info

1. `man 3 readline`
2. `man 3 history`
3. RTFM
4. RTFM
5. RTFM
6. RTFM
7. RTFM

# More Info

1. `man 3 readline`
2. `man 3 history`
3. RTFM
4. RTFM
5. RTFM
6. RTFM
7. RTFM

# More Info

1. `man 3 readline`
2. `man 3 history`
3. RTFM
4. RTFM
5. RTFM
6. RTFM
7. RTFM

# More Info

1. `man 3 readline`
2. `man 3 history`
3. RTFM
4. RTFM
5. RTFM
6. RTFM
7. RTFM

# More Info

1. `man 3 readline`
2. `man 3 history`
3. RTFM
4. RTFM
5. RTFM
6. RTFM
7. RTFM

# More Info

1. `man 3 readline`
2. `man 3 history`
3. RTFM
4. RTFM
5. RTFM
6. RTFM
7. RTFM