

# Linux Command Line: Crash-Course Tutorial

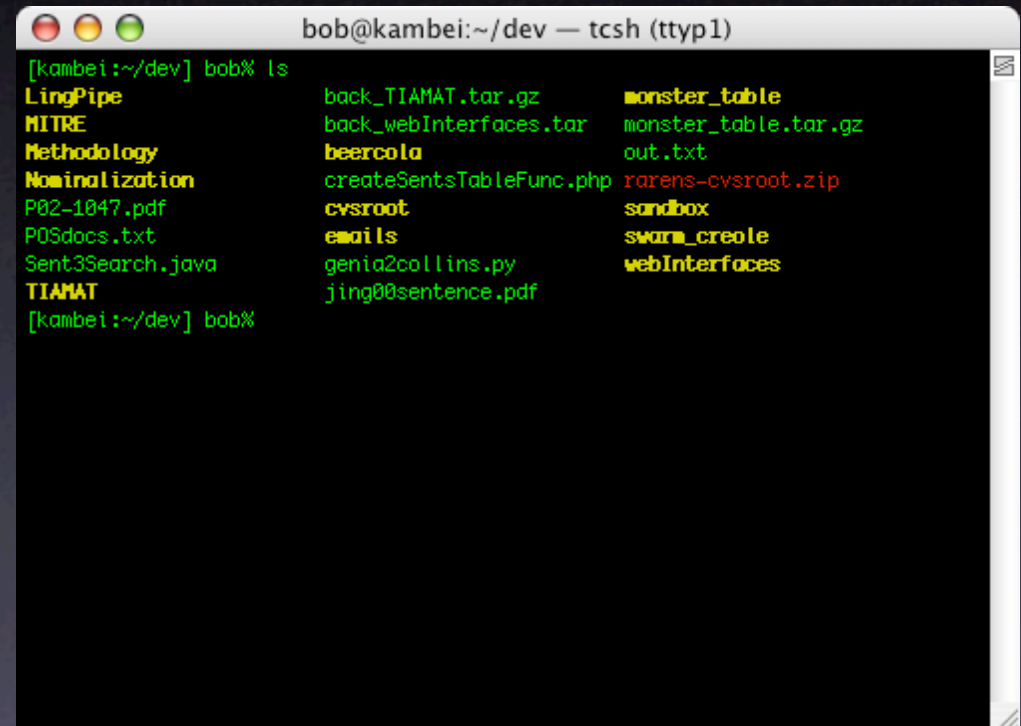
By Bob Arens

([rarens@cs.uiowa.edu](mailto:rarens@cs.uiowa.edu))

for the University of Iowa ACM

# Why use the command line?

- Speed
- Versatility
- Power
- Availability



```
bob@kambei:~/dev — tcsh (tty1)
[kambei:~/dev] bob% ls
LingPipe          back_TIAMAT.tar.gz  monster_table
NITRE             back_webInterfaces.tar  monster_table.tar.gz
Methodology       beercola            out.txt
Nominalization    createSentsTableFunc.php  rarens-cvsroot.zip
P02-1047.pdf      cvsroot             sandbox
POSdocs.txt       emails              swarn_create
Sent3Search.java  genia2collins.py    webInterfaces
TIAMAT            jing00sentence.pdf

[kambei:~/dev] bob%
```

# Where is the terminal?

- Menu - System Tools - Terminal

Find and open it now

# Getting Help

- Internet! Google is your friend.
- `man`: stands for “manual”
  - `man <program name>`
  - For example...
    - `man ls`
    - `man pwd`

# Command-line Options

- Single commands can do many different things
  - `ls` vs. `ls -a`
- Use `man` to find out more options

# File System Commands

- `ls`: list files in a directory
  - Try `ls -a`, `ls -al`
- `pwd`: print working directory
- `cd <directory>`: change directory
- `mkdir <name>`: make directory

# Special Directory Names

- `~/` : Your home directory
  - `'cd '` with no directory name brings you here
- `/` : Root (top level) directory
- `.` (one period) : Current directory
- `..` (two periods) : Parent directory
- `-` : Directory last visited

# Finding Things

- Absolute vs. relative paths
  - Relative is in relation to where you are now
  - Absolute is in relation to the root directory

# For example...

Starting from “home”

This command...	... takes you here
<code>cd foo/foo2</code>	<code>/home/foo/foo2</code>
<code>cd ../foo1</code>	<code>/home/foo/foo1</code>
<code>cd -</code>	<code>/home/foo/foo2</code>
<code>cd /foo</code>	<code>/foo</code>
<code>cd ~/bar</code>	<code>/home/bar</code>



# Excercise:

## Make some directories

- `cd`
- `mkdir foo`
- `mkdir foo/foo1`
- `cd foo`
- `mkdir foo2`
- `ls`

# Useful Shortcuts

- Tab completion
  - Type the first few letters, then the Tab key
  - Looks for programs, files, directories
- Command history
  - Up arrow recalls previous commands

# Working with Files

- `touch <file>`: Makes a new file
- `cp <file> <new file>`: Copies a file
- `mv <old file> <new file>`:  
Moves and renames a file
- `rm <file>`: Removes a file
- `rmdir <directory>`: Removes an empty directory

# Working with Files

- Can use both absolute and relative path names
  - `cp ~/file .`
  - `mv file foo/`
    - Moves the file into directory
  - `mv file /home/foo/new_file`
    - Moves and renames file

# Exercise:

## Play with some files

- `cd`
- `touch file`
- `cd foo`
- `cp ../file .`
- `cp file foo1/bar`
- `mv file foo2/bar`

# Exercise:

## More playing with files

- `cd`
- `cd foo`
- `rmdir foo1` (what happens?)
- `rm foo1/bar`
- `rmdir foo1`
- `rm -r foo2` (what happens?)

# Process Control

- Running programs are “processes”
  - Anything you do on the command line is a process
- `ps` : List user processes
  - Lots of info; username, process id, etc.
  - Many options! Look at `man ps`

# Process Control:

## Make something stop

- Suspend
  - CTRL-Z
  - fg: Bring it back
- Kill
  - `kill <process id>`
  - Use `ps` to find it (only kill yours!)

# Input / Output (I/O)

- Commands take input from user, output to screen through three common “streams”
  - `STDIN`: Input, usually the keyboard
  - `STDOUT`: Standard output to screen
  - `STDERR`: Error output to screen

# I/O Redirection

- `process > file` : Put output from `STDOUT` into new file (will overwrite files)
- `process >> file` : Add output from `STDOUT` to file
- `process_1 | process_2` : Use output to `STDOUT` of first process as input to `STDIN` for second process

# cat & grep

- `cat <file>`: Display contents of `file`
- `grep <pattern> <file>`: Find a line of text in a file
  - `pattern` is what you're looking for
    - Also called "regular expression"
  - If `file` is omitted, looks at `STDIN`

# Wildcards

- `*` : Pattern which means “match anything”
- `ls *.java` : List all files ending in `.java`
- `rm *.class` : Remove all files ending in `.class`
- Patterns are very powerful! Look for “Regular Expressions” in `man grep`

# Exercise:

## Playing with redirection

- `cd ~/foo`
- `ls -al ~/ > out`
- `cat out`
- `cat out | grep foo`
- `grep ".*" out`
  - Why do we need quotation marks?

# Text Editors

- Choose an advanced editor and learn it!
  - Plenty of online documentation
- Advanced editors
  - emacs, vi
- Simpler editors - not as powerful!
  - gedit, nedit, jedit, nano, pico

# zipped and tarred

- `.gz` : File compressed with `gzip`
- `.tar` : Files packed together with `tar`
  - `.tar.gz` : Files packed and compressed
- `.zip` : Files compressed with `zip`

# zipped and tarred: creating them

- `tar -cf file.tar <directory>`
- `gzip file`
  - New file: `file.gz`
- `tar -czf file.tar.gz <dir>`
- `zip file.zip <file1> ...`

# zipped and tarred: opening them

- `tar -xf file.tar`
- `gunzip file.gz`
- `tar -xzf file.tar.gz`
- `unzip file.zip`
- Always unzip and untar inside a new directory