

Unix Basics

Benjamin S. Skrainka
Harris School of Public Policy
University of Chicago

July 19, 2012

Overview

We will cover basic Unix survival skills:

Why you need some Unix in your life

How to get some Unix in your life

Basic commands

(Free) tools you can't live without

Why you need some Unix in your life

Unix/Linux will make you more productive:

- ▶ Simple yet powerful commands to solve common problems
- ▶ 'Building blocks' + 'glue' to quickly build new tools
- ▶ Command line or GUI interface
- ▶ Access to vast amounts of Open Source or free software
- ▶ Forgiving programming environment
- ▶ Exceptionally robust architecture
- ▶ The lingua franca for scientific and high throughput computing

Design Philosophy

The core of Unix's design philosophy is:

- ▶ A command/program should do one thing only, but do it well (building blocks)
- ▶ Commands should be easy to string together to perform more complex operations (glue)
- ▶ I.e. 'filters and pipes'
- ▶ Smaller kernel than Windows
 - ▶ Greater reliability and extensibility
 - ▶ Better performance
 - ▶ Software errors are less likely to be catastrophic
- ▶ Written by smart people for smart people. . .

How to get some Unix in your life

There are several ways to get some Unix:

- ▶ PC hardware:
 - ▶ Download cygwin (www.cygwin.com)
 - ▶ Install Linux
 - ▶ Can dual-boot Linux and Windows
 - ▶ Ubuntu is a popular distribution
- ▶ OS/X
 - ▶ Has Unix-style kernel underneath user interface
 - ▶ Install XCode (developer.apple.com)
 - ▶ Install MacPorts (www.macports.org)
 - ▶ Install desired packages with `port` command
- ▶ Use a virtual machine: VMWare Fusion, Parallels, VirtualBox (Free)

Overview

Let's look at the basic survival skills needed on Unix:

- ▶ The Shell
- ▶ Getting Help
- ▶ Configuration
- ▶ Files and Directories
- ▶ Editing
- ▶ Job Control

The Shell

On a Unix/Linux machine, you interact with a shell:

- ▶ Connect to a shell:
 - ▶ via NX-Client, VNC, etc. → graphical
 - ▶ via Terminal, xterm, ssh, etc. → command line
- ▶ The shell is a program:
 - ▶ Reads your commands and executes them
 - ▶ Is also a programming language – you can write scripts to automate common commands
- ▶ Several shells are available: bash, sh, csh, ...
- ▶ Bash is now the most popular
- ▶ Can configure via `~/.bashrc`, `~/.profile`, and `~/.bash_profile`

Help

To get help:

- ▶ Use the man command:

```
man man
```

```
man ls
```

```
man -k edit
```

- ▶ Navigation: space, /, f, b, ...

- ▶ Use GUI help command, if supported
- ▶ Google
- ▶ [StackOverflow](#)
- ▶ O'Reilly books (www.ora.com)
- ▶ [A Practical Guide to Linux, Editors, and Shell Programming](#) by Mark G. Sobell

Files and Directories

Navigate by specifying filenames and directories (folders):

`ls` *[dir]* list the contents of a directory

`pwd` display current directory

`cd` *dir* change directory

`mkdir` *dir* make directory

`rmdir` *dir* remove directory

`rm` *file* delete a file (N.B. there is no trash!)

`rm -rf *` nuke everything

`mv` *from to* move/rename a file or directory

`find` traverse a directory tree and execute commands

Conventions

There are a couple conventions to be aware of:

- ▶ Special characters in filenames:
 - . Current directory
 - .. Parent directory
 - * Greedy matching of all characters in a name
 - ~ Your HOME directory (cd without arguments goes to ~)
- ▶ Dotfiles:
 - ▶ Invisible unless you use `ls -a`
 - ▶ Used for configuration
 - ▶ `.bashrc`, `.profile`, `.login`, and application-specific files
 - ▶ The place to define your own commands with `alias`

Unix vs. Windows vs. Mac Confusion

Unfortunately, there is often confusion when moving between Windows/DOS and Unix/Linux/Mac:

- ▶ Different separators in pathnames:

***nix:** /path/to/my/file.txt

Mac: /path/to/my/file.txt

Windows: c:\path\to\my\file.txt

But, ‘\’ is used to escape special characters such as ‘\n’ for line feed or ‘\\’ for ‘\’...

- ▶ Different conventions for line termination:

***nix:** LF

Mac: CR

Windows: CR+LF

May need to convert text files when changing platforms with `dos2unix`, `unix2dos`, `sed`, or `tr`.

UTF-8

Beware of UTF-8 and other i18n (internationalization) encodings because bash expects text to be just ASCII characters:

- ▶ Often used with Asian languages
- ▶ Can be a problem if cutting and pasting code from email or the web
- ▶ E.g., there are several different versions of some characters such as the apostrophe
- ▶ Bottom line: if the text looks correct but is producing weird error messages then retype everything...
- ▶ Can convert using `iconv`

Configuration

Most Unix programs can be configured:

- ▶ Configuration information is stored in the application's dotfile in your login directory (~)
- ▶ Specify PATH, LD_LIBRARY_PATH, EDITOR, and other environment variables
- ▶ Define alias:

```
alias h='history'  
alias l='ls -F'  
alias ll='ls -lFh'  
alias py='python2.7'  
alias xsede='ssh skrainka@blacklight.psc.teragrid.org'
```

Environment Variables

Use environment variables specify configuration or state information:

- ▶ Typically set in `~/.bashrc` or `~/.profile` for bash
- ▶ Access by prefacing name with '\$'
- ▶ Key environment variables are:
 - ▶ `PATH` : a list of directories the shell searches for commands
 - ▶ `LD_LIBRARY_PATH` : a list of directories which the operating system searches for shared libraries needed by programs
- ▶ Other applications have their own configuration files or directories, e.g. `.vimrc` for vim, `.matlab` for MATLAB, and `.ssh` for ssh.
- ▶ Display with `env` or `echo $VAR_NAME`
- ▶ See manual for more complex operations

History

Unix has a sophisticated history facility:

`history` list recent commands

`!n` reexecute n -th command

`!cmd` reexecute most recent command which started with string *cmd*

`^P` scroll backwards through history (can use arrow keys...)

`!cmd:p` load most recent command starting with *cmd* onto command line (for editing or execution)

More sophisticated manipulations are possible

Looking at Stuff

Unix has many handy commands for manipulating text files:

`less` Page through a file

`grep` Search for a token

`cat` Concatenate files (or dump them to the screen)

`head` Cat the top of a file

`tail` Cat the end of a file

`wc` Display number of characters, words, and/or lines

`cmp` Test if two files are the same (can use on binary files)

`diff` Show differences between two files

`sum/md5/md5sum` Compute checksum (can use on binary files)

Editing

Traditional editors are:

- ▶ vi
- ▶ emacs
- ▶ Supported everywhere!

Other options (if installed):

- ▶ nano or pico
- ▶ jEdit (Download from www.jedit.org)

Learning a programming editor will increase your productivity!

Permissions

Unix-style permissions are confusing to the uninitiated:

```
% ls -la
total 440
drwxr-xr-x 23 bss staff 782 24 Jul 22:05 .
drwxr-xr-x 11 bss staff 374 24 Jul 22:06 ..
drwxr-xr-x  8 bss staff 272 27 Jul 18:25 .svn
-rw-r--r--@ 1 bss staff 12655 24 Jul 22:06 BasicDriver.m
-rw-r--r--  1 bss staff 16128 24 Jul 21:54 BasicDriver.m~
...
```

[d|-] directory or not

[rwx] permissions are grouped according to social distance:

- ▶ *user, group, and world*
- ▶ Specify r, w, and x (Octal: 4, 2, 1)

chmod: use to change permissions: `chmod 755 myFile.m`

Remote Login

To connect to a remote machine use the secure shell protocol:

- ▶ Unix, Linux, or OS/X:
 - ▶ `ssh -Y YourLoginName@htc.uni.edu`
 - ▶ Can also use `sftp` and `scp`
- ▶ Windows:
 - ▶ Download PuTTY
 - ▶ Create a connection via GUI
 - ▶ May need to configure colors
- ▶ Uses encryption to provide a secure connection
- ▶ Do not use `rlogin`, `telnet`, or `ftp` (unless anonymous) which are not secure!!!

Building More Complex Commands

Unix provides tools to link 'atomic' commands together into more complex commands:

- ▶ IO Redirection: >, <, <<
- ▶ Pipe: |
- ▶ Shell scripts
- ▶ Regular Expressions
- ▶ Think of commands as filters in a pipeline joined by pipes. . .

Example:

```
egrep -e '^ [0-9]\{1,2\}[a-z]' Data.txt | sort > out.txt
```

Pattern Matching

Most Unix tools support *Regular Expressions*:

- ▶ Powerful, compact, and often cryptic language for specifying patterns
- ▶ Permits sophisticated searching via `egrep`, `vi`, `emacs`
- ▶ Permits sophisticated editing via `vi`, `emacs`, `sed`, `awk`, `perl`, `python`, etc.
- ▶ Can capture parts of a pattern and manipulate
- ▶ Simple example to reverse columns separated by '=':

```
sed 's/\(.*\)=(.*)/\2 = \1/' SomeFile.txt
```

Job Control

Processes are organized in a hierarchical manner:

- ▶ Every process has a *parent*
- ▶ The parent forks and execs a *child* process
- ▶ Kill the parent, and all its children also die
- ▶ Reference with a ***Process ID***
- ▶ Dead children are *reaped*...

Process Control

Basic process control commands include:

`top` List processes consuming most resources

`ps` Get information about processes

`cmd &` Run `cmd` in background process

`jobs` List process running in background

`kill -9 PID` Terminate a process

`kill %JobID` Terminate using job ID

`xkill` Terminate a process graphically

`users` Who is logged in (variants: `w` and `who`)

`uptime` How long since last reboot + load average

(Free) tools you can't live without

Unix rules for data janitorial activities such as process text, extracting information from a stream of output, automating analysis of log files, etc.

- ▶ Version control: `git`
- ▶ Stream Editors: `sed`, `awk`, etc.
- ▶ Python
- ▶ R
- ▶ `make`
- ▶ Eclipse (PhoTran, C/C++, Java)
- ▶ `doxygen`

Listing 1: Text Extraction for NEOS Server: sed + bash

```
#!/bin/bash
INFILE=neosOutput.txt

for varName in V VK1 VK2 VK3 MIU HELPC HELPQ
do
    sed -n "/${varName}_\[/,/;/P" $INFILE > out.${varName}.txt
done
```

Listing 2: Text Extraction for NEOS Server: Python

```
#!/usr/bin/env python
import re
import sys
import string

szInFile = sys.argv[ 1 ]
szOutDir = sys.argv[ 2 ]

vTokens = [ 'V', 'VK1', 'VK2', 'VK3', 'MIU' ]

# Load NEOS input file
fln = open( szInFile )
vText = fln.read( )
fln.close()

for szToken in vTokens:
    pat = re.compile( szToken + "\[[^;]*;", re.DOTALL )
    tgt = pat.search( vText )
    ixStart = tgt.start()
    ixEnd = tgt.end()
    f = open( szOutDir + '/' + szToken + '.out', 'w' )
    f.write( vText[ ixStart:ixEnd ] )
    f.close()
```

Comparing Data Files

```
#!/usr/bin/env python
"""
isApprox.py - test approximate equality of data
"""
import numpy as np
import sys
# Setup
if 3 != len( sys.argv ) :
    print 'Syntax error: isApprox.py file1 file2'
    sys.exit( -1 )

szFile1 = sys.argv[ 1 ]
szFile2 = sys.argv[ 2 ]
m1 = np.loadtxt( szFile1 )
m2 = np.loadtxt( szFile2 )
```

Comparing Data Files

```
# Compare data
if m1.ndim != m2.ndim :
    print 'Matrices are not conformable.'
    sys.exit( -1 )
if m1.shape[ 0 ] != m2.shape[ 0 ] :
    print 'Error: different numbers of rows.'
    sys.exit( -1 )
if 2 == m1.ndim :
    if m1.shape[ 1 ] != m2.shape[ 1 ] :
        print 'Error : different numbers of columns.'
        sys.exit( -1 )
print 'Norm(diff): ', np.linalg.norm( m1 - m2, ord=2 )
print 'max abs diff : ', np.max( np.abs( m1 - m2 ) )
```