Unix workshop: The Basics

Zaki Mughal

Computational Biomedicine Lab
University of Houston

2013 Nov 13

Slides are at `http://bit.ly/csgsa_unix_f2013`.

Introduction

Intro

- Unix : operating system developed at Bell Labs in 1969

Intro

- Unix : operating system developed at Bell Labs in 1969
- written in C and assembly

Intro

- Unix : operating system developed at Bell Labs in 1969
- written in C and assembly

  portability

Intro

- Unix : operating system developed at Bell Labs in 1969
- written in C and assembly

  portability
- response to Multics OS project

Intro

- Unix : operating system developed at Bell Labs in 1969
- written in C and assembly
  - portability
- response to Multics OS project
- developed officially for text processing

Intro

- Unix : operating system developed at Bell Labs in 1969
- written in C and assembly
  portability
- response to Multics OS project
- developed officially for text processing (but really games)

Intro

- Unix : operating system developed at Bell Labs in 1969
- written in C and assembly
  portability
- response to Multics OS project
- developed officially for text processing (but really games)
- multi-user (time-sharing)

Intro

- Unix : operating system developed at Bell Labs in 1969
- written in C and assembly
  portability
- response to Multics OS project
- developed officially for text processing (but really games)
- multi-user (time-sharing)
- distributed to anyone that wanted it

Later

- BSD $\rightarrow$ FreeBSD, NetBSD, OpenBSD

Later

- BSD $\rightarrow$ FreeBSD, NetBSD, OpenBSD
- HP-UX, Solaris, AIX, Irix, Xenix

Later

- BSD $\rightarrow$ FreeBSD, NetBSD, OpenBSD
- HP-UX, Solaris, AIX, Irix, Xenix
- *Linux*

Later

- BSD $\rightarrow$ FreeBSD, NetBSD, OpenBSD
- HP-UX, Solaris, AIX, Irix, Xenix
- *Linux*
- Mac OS X

Later

- BSD $\rightarrow$ FreeBSD, NetBSD, OpenBSD
- HP-UX, Solaris, AIX, Irix, Xenix
- *Linux*
- Mac OS X

POSIX, SUS — standardisation of API and environment between Unix systems

Concepts

- (almost) everything is a file
- single tree (no concepts of multiple drives)
- top is / (root directory)

Standard directories (where's the thing for that thing)

- /bin — binaries essential for the system (used by everyone)
- /sbin — binaries for managing the system (usually for sysadmin)
- /lib — libraries used for /bin, /sbin

Standard directories (where's the thing for that thing)

- /bin — binaries essential for the system (used by everyone)
- /sbin — binaries for managing the system (usually for sysadmin)
- /lib — libraries used for /bin, /sbin
- /usr/bin, /usr/sbin, /usr/lib — same as above, but not system critical

Standard directories (where's the thing for that thing)

- /bin — binaries essential for the system (used by everyone)
- /sbin — binaries for managing the system (usually for sysadmin)
- /lib — libraries used for /bin, /sbin
- /usr/bin, /usr/sbin, /usr/lib — same as above, but not system critical
- /usr/include — header files (.h)

Standard directories (where's the thing for that thing)

- /bin — binaries essential for the system (used by everyone)
- /sbin — binaries for managing the system (usually for sysadmin)
- /lib — libraries used for /bin, /sbin
- /usr/bin, /usr/sbin, /usr/lib — same as above, but not system critical
- /usr/include — header files (.h)
- /tmp — temporary storage (cleaned up at boot)
- /dev — device files (tty0, lp0, hda)
- /mnt — other filesystems (another partition, hard drive, CD-rom)

Standard directories (where's the thing for that thing)

- /bin — binaries essential for the system (used by everyone)
- /sbin — binaries for managing the system (usually for sysadmin)
- /lib — libraries used for /bin, /sbin
- /usr/bin, /usr/sbin, /usr/lib — same as above, but not system critical
- /usr/include — header files (.h)
- /tmp — temporary storage (cleaned up at boot)
- /dev — device files (tty0, lp0, hda)
- /mnt — other filesystems (another partition, hard drive, CD-rom)
- /home — home directories of users

Standard directories (where's the thing for that thing)

- /bin — binaries essential for the system (used by everyone)
- /sbin — binaries for managing the system (usually for sysadmin)
- /lib — libraries used for /bin, /sbin
- /usr/bin, /usr/sbin, /usr/lib — same as above, but not system critical
- /usr/include — header files (.h)
- /tmp — temporary storage (cleaned up at boot)
- /dev — device files (tty0, lp0, hda)
- /mnt — other filesystems (another partition, hard drive, CD-rom)
- /home — home directories of users

  /home/user1

  /home/user2

  ...

. and ..

- each directory has two filenames: . and ..

. and ..

- each directory has two filenames: . and ..
- . is the current directory

. and ..

- each directory has two filenames: `.` and `..`
- `.` is the current directory
- `..` is the directory above the current directory

`.` and `..`

- each directory has two filenames: `.` and `..`
- `.` is the current directory
- `..` is the directory above the current directory

e.g. `/usr/lib/.` is another name for `/usr/lib`

`.` and `..`

- each directory has two filenames: `.` and `..`
- `.` is the current directory
- `..` is the directory above the current directory

e.g. `/usr/lib/.` is another name for `/usr/lib`
and `/usr/lib/..` is another name for `/usr`

The Prompt

- command line interface
- scriptable — automation
- short commands for fast entry
- managed by a *shell*
  (on Linux, we'll be using a shell called `bash`)

The Prompt

- command line interface
- scriptable — automation
- short commands for fast entry
- managed by a *shell*
  (on Linux, we'll be using a shell called `bash`)
- bash has tab-complete

The Prompt

- command line interface
- scriptable — automation
- short commands for fast entry
- managed by a *shell*
  (on Linux, we'll be using a shell called `bash`)
- bash has tab-complete — please use it :-)

Filesystem commands

echo

echoes arguments to `stdout`

this is built-in to bash, but `/bin/echo` also exists

ls

```
$ ls            # what's in the current directory?
test.c          test.h
note            contrib/
```

ls

```
$ ls          # what's in the current directory?
test.c        test.h
note          contrib/
$ ls contrib # what's in contrib/ ?
data.txt
$
```

ls

```
$ ls            # what's in the current directory?
test.c          test.h
note            contrib/
$ ls contrib # what's in contrib/ ?
data.txt
$
```

list files and directories

options:

- -a : all files (including hidden files)
- -l : long format (shows extra information)

hidden files

In Unix, hidden files are any files that begin with a dot.
e.g.

- .
- ..
- .bashrc
- .vimrc
- .git/

usually used for configuration

cd and pwd

```
$ cd contrib   # change directory
$ ls
data.txt
$ pwd          # path to working directory
/home/user1/project/contrib
$
```

mkdir

```
$ cd ..  # go back up to /home/user1/project
```

mkdir

```
$ cd ..  # go back up to /home/user1/project
$ mkdir data
$ cd data
```

mkdir

```
$ cd ..  # go back up to /home/user1/project
$ mkdir data
$ cd data
$ ls
$ # it's empty
```

make a directory

mkdir

```
$ cd ..  # go back up to /home/user1/project
$ mkdir data
$ cd data
$ ls
$ # it's empty
```

make a directory

options:

- -p : make any parents as well

  `mkdir -p a/b/c`

touch, redirection, cat

```
$ touch more_data.txt
```

touch, redirection, cat

```
$ touch more_data.txt
$ ls
more_data.txt
```

touch, redirection, cat

```
$ touch more_data.txt
$ ls
more_data.txt
$ cat more_data.txt # new file is empty
```

touch, redirection, cat

```
$ touch more_data.txt
$ ls
more_data.txt
$ cat more_data.txt # new file is empty
$ echo "Some data" > more_data.txt
```

touch, redirection, cat

```
$ touch more_data.txt
$ ls
more_data.txt
$ cat more_data.txt # new file is empty
$ echo "Some data" > more_data.txt
$ cat more_data.txt
Some data
```

touch, redirection, cat

```
$ touch more_data.txt
$ ls
more_data.txt
$ cat more_data.txt # new file is empty
$ echo "Some data" > more_data.txt
$ cat more_data.txt
Some data
$ echo "Extra data" > extra_data.txt
```

touch, redirection, cat

```
$ touch more_data.txt
$ ls
more_data.txt
$ cat more_data.txt # new file is empty
$ echo "Some data" > more_data.txt
$ cat more_data.txt
Some data
$ echo "Extra data" > extra_data.txt
$ ls
more_data.txt        extra_data.txt
```

touch, redirection, cat

```
$ touch more_data.txt
$ ls
more_data.txt
$ cat more_data.txt # new file is empty
$ echo "Some data" > more_data.txt
$ cat more_data.txt
Some data
$ echo "Extra data" > extra_data.txt
$ ls
more_data.txt          extra_data.txt
```

touch creates a file if it doesn't exist or updates timestamp of file

touch, redirection, cat

```
$ touch more_data.txt
$ ls
more_data.txt
$ cat more_data.txt # new file is empty
$ echo "Some data" > more_data.txt
$ cat more_data.txt
Some data
$ echo "Extra data" > extra_data.txt
$ ls
more_data.txt          extra_data.txt
```

touch creates a file if it doesn't exist or updates timestamp of file
cat outputs contents of files

touch, redirection, cat

```
$ touch more_data.txt
$ ls
more_data.txt
$ cat more_data.txt # new file is empty
$ echo "Some data" > more_data.txt
$ cat more_data.txt
Some data
$ echo "Extra data" > extra_data.txt
$ ls
more_data.txt        extra_data.txt
```

`touch` creates a file if it doesn't exist or updates timestamp of file
`cat` outputs contents of files (Be careful with binary files. They can mess up your terminal. Use `reset` to fix that.)

cp, mv

```
$ cp more_data.txt more_data.txt.old
# copy file
```

cp, mv

```
$ cp more_data.txt more_data.txt.old
# copy file
$ ls
more_data.txt    extra_data.txt     more_data.txt.old
```

cp, mv

```
$ cp more_data.txt more_data.txt.old
# copy file
$ ls
more_data.txt    extra_data.txt     more_data.txt.old
$ mv extra_data.txt more_data.txt.old
# move/rename file
```

cp, mv

```
$ cp more_data.txt more_data.txt.old
# copy file
$ ls
more_data.txt    extra_data.txt     more_data.txt.old
$ mv extra_data.txt more_data.txt.old
# move/rename file
$ ls
more_data.txt    more_data.txt.old
```

cp, mv

```
$ cp more_data.txt more_data.txt.old
# copy file
$ ls
more_data.txt    extra_data.txt     more_data.txt.old
$ mv extra_data.txt more_data.txt.old
# move/rename file
$ ls
more_data.txt    more_data.txt.old
```

options for cp:

- -p : preserve timestamp
- -r, -R : recursively copy (needed for directories)

My own habit for copying directories is to use `cp -puvR`.

rm

```
$ cd ..
```

rm

```
$ cd ..
$ ls
test.c      test.h
note        contrib/
data/
```

rm

```
$ cd ..
$ ls
test.c          test.h
note            contrib/
data/
$ rm note
```

rm

```
$ cd ..
$ ls
test.c      test.h
note        contrib/
data/
$ rm note
$ ls
test.c      test.h
contrib/    data/
```

rm

rm

```
$ rm data/*
# use wildcard to remove all files inside
$ ls data
$ rmdir data # remove empty directory
$ ls
test.c          test.h          contrib/
```

rm

```
$ rm data/*
# use wildcard to remove all files inside
$ ls data
$ rmdir data # remove empty directory
$ ls
test.c      test.h      contrib/
$ rm -R contrib
```

rm

```
$ rm data/*
# use wildcard to remove all files inside
$ ls data
$ rmdir data # remove empty directory
$ ls
test.c        test.h        contrib/
$ rm -R contrib
$ ls
test.c        test.h
```

rm

```
$ rm data/*
# use wildcard to remove all files inside
$ ls data
$ rmdir data # remove empty directory
$ ls
test.c        test.h        contrib/
$ rm -R contrib
$ ls
test.c        test.h
```

options for `rm`:

- -r, -R : recursively delete (needed for directories)

I use `-R` instead of `-r` because it stands out more. There is no built-in trash can in Unix.

du, df

```
$ du
[... outputs file sizes
 for the current directory ... ]
$ df
[... outputs disk info ... ]
```

Documentation

man

```
$ man ls
[ documentation for ls ]
$ man printf
[ documentation for the printf command ]
$ man 3 printf
[ documentation for printf() in stdio.h ]
$ man man
```

man

```
$ man ls
[ documentation for ls ]
$ man printf
[ documentation for the printf command ]
$ man 3 printf
[ documentation for printf() in stdio.h ]
$ man man
```

man pages have a standard layout to make navigating them easier

man

```
$ man ls
[ documentation for ls ]
$ man printf
[ documentation for the printf command ]
$ man 3 printf
[ documentation for printf() in stdio.h ]
$ man man
```

man pages have a standard layout to make navigating them easier

Some commands are shell built-ins. Use `help` to see the documentation for these

Conventions

```
$ ls -a      # short option
```

Conventions

```
$ ls -a     # short option
$ ls -la    # short options can be combined
```

Conventions

```
$ ls -a     # short option
$ ls -la    # short options can be combined
$ ls -l -a  # or separate
```

Conventions

```
$ ls -a     # short option
$ ls -la    # short options can be combined
$ ls -l -a  # or separate
$ ls --all  # long options have two hyphens
```

Conventions

```
$ ls -a      # short option
$ ls -la     # short options can be combined
$ ls -l -a   # or separate
$ ls --all   # long options have two hyphens
$ ls --all --full-time # must be separate
```

Access control

Users

- root user (admin)

Users

- root user (admin)
- su, sudo (change user)

Users

- root user (admin)
- su, sudo (change user)
- whoami (who is the current user?)

Users

- root user (admin)
- su, sudo (change user)
- whoami (who is the current user?)
- w (who is logged in?)

Groups

- `groups` (get the groups of a user)

Permissions

- Each file is owned by a single user and a single group

Permissions

- Each file is owned by a single user and a single group
- shown in `ls -l` listing

Permissions

- Each file is owned by a single user and a single group
- shown in `ls -l` listing
- we can set the group that a file belongs to

```
$ chgrp mygroup project.txt
```

Permissions

- Three kinds of permissions
  - r — read
  - w — write
  - x — execute

Permissions

- Three kinds of permissions
  - r — read
  - w — write
  - x — execute
- we can set these settings for the owner of the file, the group the file is in, and anyone else

Permissions

```
$ ls -l this_needs_to_execute
-r-------- 1 user1 group1  this_needs_to_execute
```

Permissions

```
$ ls -l this_needs_to_execute
-r-------- 1 user1 group1  this_needs_to_execute
$ chmod u+x this_needs_to_execute  # user
```

Permissions

```
$ ls -l this_needs_to_execute
-r-------- 1 user1 group1  this_needs_to_execute
$ chmod u+x this_needs_to_execute  # user
$ chmod g+rx this_needs_to_execute # group
```

Permissions

```
$ ls -l this_needs_to_execute
-r-------- 1 user1 group1  this_needs_to_execute
$ chmod u+x this_needs_to_execute  # user
$ chmod g+rx this_needs_to_execute # group
$ chmod o+rx this_needs_to_execute # other
```

Permissions

```
$ ls -l this_needs_to_execute
-r-------- 1 user1 group1  this_needs_to_execute
$ chmod u+x this_needs_to_execute  # user
$ chmod g+rx this_needs_to_execute # group
$ chmod o+rx this_needs_to_execute # other
$ ls -l this_needs_to_execute
-r-xr-xr-x 1 user1 group1  this_needs_to_execute
```

Next time

- processes
- more on I/O redirection
- screen (terminal multiplexer)
- advanced scripting
- network