The Unix Command Line

The Shell

- The shell is a command line interpreter
- It reads commands and executes them
- Some commands are builtin (part of the shell program)
- Some commands are external programs
- The shell is a program (usually written in C).

Shell Functions

- Provides a way for a user to interact with an operating system
- Is a scripting language

Types of Shells

GUI shells

- Easy for novices to use
- Limited in power

Command line interpreters

- Steeper learning curve
- Far more capabilities
- Allows automation of repetitive tasks

Various Shells

- The Thompson shell (sh) Ken Thomson - Bell Labs
- The Bourne Shell (sh) Stephen Bourne - Bell Labs
- The Bourne-Again Shell (bash) Brian Fox - GNU Project
- The Dash Shell (dash) Herbert Xu - Red Hat
- The C Shell (csh) Bill Joy - Berkeley
- The Tenex C Shell (tcsh) Ken Greer - Carnegie Mellon
- Bash is the default user shell for Ubuntu (and many others) Linux
- Dash is the default shell for executing scripts in Ubuntu

Simple Command Line Syntax (1)

- *command options additional-arguments*
- Example:

```
ls -a -l foobar
```

Single Letter Options

- Begin with a single hyphen
- Can be specified together or separately

```
ls -a -l foobar
ls -al foobar
```

- Not all commands follow this pattern

Simple Command Line Syntax (2)

Long Options

- Begin with two hyphens
- Example

```
ls --all --reverse foobar
```

- Not all commands follow this pattern
- `cc` uses a single hyphen for long arguments
- As a result

```
cc -SO file
```

does not work. It must be

```
cc -S -O prog.c
```

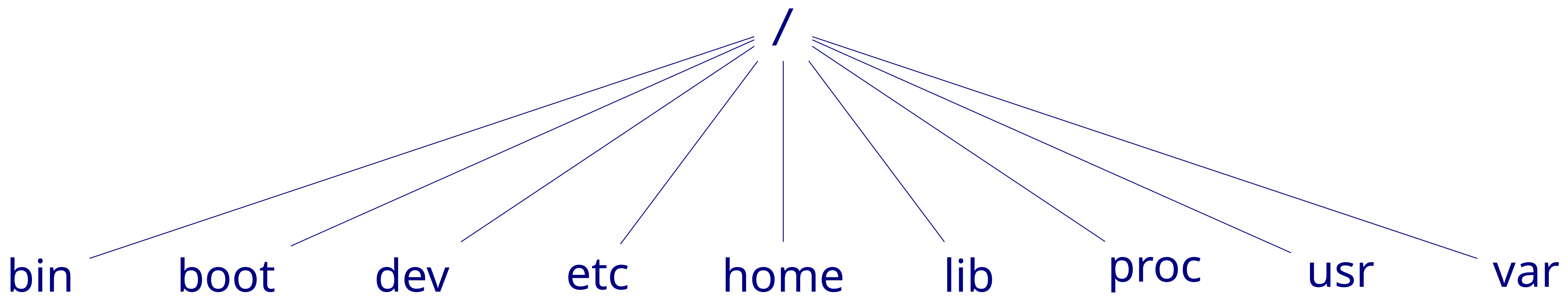
Simple Command Line Syntax (3)

- Additional arguments are frequently, but not always, file names.
- Sometimes options have associated arguments.

```
head -n 20 foobar
head --lines=20 foobar
```

Directories

- Files are organized into directories (similar to folders)
- The file system is hierarchical (tree-like)
- All files are descendants of the root directory (called /).
- Typical hierarchy:



Current Directory

- Each user has a home directory that they own.
- The shell keeps trace of a current (working) directory.
- When you first log in, the current directory is your home directory.
- pwd prints the name of the current directory.
- `ls` with no arguments lists files in the current directory.

Pathnames

- Path names can be absolute or relative.
- Absolute path names begin with a (forward) slash.
- An absolute path contains the name of every directory above the file in hierarchy.
- Example: `/home/mathcs/courses/cs246/test-tux`
- A relative path name does not start with a slash.
- A relative path contains the name of every directory between the current directory and the file.
- Example: `cs246/homework1/homework12.c`

Special Directory Names

- `.` always refers to the current directory.
- `..` always refers to the parent of the current directory.
- ~ always refers to the user’s home directory.

Standard Streams

- When a command executes, it starts with 3 standard streams.
- All 3 streams are connected to the terminal by default.
  - standard input (stdin) - normal terminal input
  - standard output (stdout) - normal terminal output
  - standard error (stderr) - error messages

Redirecting Output

- Redirect standard output to a file like this:

```
ls > outfile
```

- This redirects standard output, but messages sent to standard error still go to the terminal.
- Redirecting standard output and standard error:

```
ls >& outfile
```

Redirecting Input

- Redirect standard input to come from a file like this:

```
cat -n < infile
```

- Most commands that take input read from standard input if no file is specified.
- Because `cat` takes file arguments, we can run it in 2 ways:

```
cat < infile
cat infile
```

- Some commands do not take file arguments.
- We must run `tr` this way:

```
tr A-Z a-z < infile
```

Redirecting Input and Output

- You can specify input and output redirection in either order.

```
cat < infile > outfile
cat > outfile < infile
cat < infile >& outfile
```

- IMPORTANT! Do not use input and output redirection for the same file. The shell will open the file for output, erasing the contents, before reading the file.

```
cat < infile > infile
```

- This makes infile an empty file, regardless of the previous contents.

Control Keys

`Ctrl-c`	Send an interrupt signal, which usually kills a program.
`Ctrl-d`	At the beginning of a line, signals end-of-file.
`Ctrl-l`	Redisplay screen.
`Ctrl-z`	Suspend a program. Resume with `fg` (foreground) or bg (background).

End of File for the Terminal

- When a command is reading from the terminal, you can specify end of file with `Ctrl-d` at the beginning of a line.
- The world’s stupidest text editor:

```
cat > outfile
...
Ctrl-d
```

- This works great if you never make a mistake.

Special Characters

- Several characters have special meanings to the shell.

```
#  |  &  ;  ! * $ < > ( )  { } [ ]
space tab newline
```

- It is best to not use any of these in file names.

Quoting

- To use a character without its special meaning, you must quote it.
- How to quote:
  - Enclose the character (or the word containing it) in single quotes: `'foo & bar'`
  - Enclose the character in double quotes: `"foo & bar"`
  - Escape the character with a backslash: `foo\ \&\ bar`
- Variable references are replaced by their values inside double quotes but not inside single quotes.

```
prompt > x=foobar
prompt > echo '$x'
$x
prompt > echo "$x"
foobar
```

Pipelines

- Pipelines allow us to combine programs to do complex tasks.
- A pipeline connects standard output for one program to standard input for another.
- Example: `less` is a program that displays text one screenful at a time. It is often used when a program produces a lot of output. If you have a directory with many files,

  ```
  ls | less
  ```

  will display the output of `ls` one screenful at a time.
- Any number of files can be combined in a pipeline.

  ```
  p1 < infile | p2 | p3 | p4 | p5 > outfile
  ```

  will run the program p1 with input from infile and filter the output through the programs p2, p3, p4, and p5, putting the final output into outfile.

Pipeline Example (1)

We will get a list of all the words appearing in a file.

- First convert uppercase to lowercase.

  ```
  tr A-Z a-z < infile
  ```

- Next convert all nonletters to newlines.

  ```
  tr A-Z a-z < infile | tr -c a-z '\n'
  ```

- Sort the output.

  ```
  tr A-Z a-z < infile | tr -c a-z '\n' | sort
  ```

Pipeline Example (2)

- Remove duplicates.

```
tr A-Z a-z < infile | tr -c a-z '\n' | sort | uniq
```

- Remove the first line.

```
tr A-Z a-z < infile | tr -c a-z '\n' | sort | uniq | tail -n +2
```

- Redirect the output to a file.

```
tr A-Z a-z < infile | tr -c a-z '\n' | sort | uniq | tail -n +2 > outfile
```

Globbing (Pathname Expansion

- The shell expands a pattern containing `*`, `?`, and `[...]` into a list of filenames matching the pattern.
  - `*` matches any string (including the empty string).
  - `?` matches any single character.
  - `[`*`list-of-characters`*`]` matches any single character that occurs in the list.
  - Do not match names starting with `.` unless the pattern starts with `.`
- This process is called globbing.

Glob Patterns (1)

Pattern	Possible Matches
foo	f
a*	names beginning with a
*.c	C programs
foobar	names containing foobar
a*n*z	names starting with a, ending with z, and containing n somewhere in the middle
*	any name
????	names of length 4
????*	names of length 4 at least 4
??.jpg	names of jpeg files of length 6

Glob Patterns (2)

Pattern	Possible Matches
[abc]*	names starting with a, b, or c
[a-z]*	names starting with a lowercase letter
[foobar]	f, o, b, a, r (not foobar!)
[^0-9]*	names that do not start with a digit
[^A-Za-z]*	names that do not start with a letter
.[^.]*	names that start with a period but the second character is not a period

Brace Expansion

- A brace expression is a list of words separated by commas enclosed in curly braces.

- The shell expands these to all possibilities. Unlike glob expressions, they are expanded whether there are matching file names or not.

- Examples:

  - `{jpg,png,gif}` expands to `jpg png gif` (not useful by itself)

  - `file1.{jpg,png,gif}` expands to `file1.jpg file1.png file1.gif`

  - `*.{jpg,png,gif}` expands to all file names ending in `.jpg`, `.png`, or `.gif`

  - `*.{java,py,c}` expands to all java, python, or c programs

  - `{a,b,c}{d,e}` expands to `ad ae bd be cd ce`

- Ranges

  - `{1..4}` expands to `1 2 3 4`

  - `{1..3}{a..d}` expands to `1a 1b 1c 1d 2a 2b 2c 2d 3a 3b 3c 3d`

  - `{1..20..3}` expands to `1 4 7 10 13 16 19`