CS 45, Lecture 2
Shell Tools

Winter 2023

Akshay Srivatsan, Ayelet Drazen, Jonathan Kula

Outline

1. What is the Shell?
2. The UNIX Shell
3. Basic Commands
4. Pipes
5. Conclusion

Outline

UNIX

- The shell (as we recognize it) began with the UNIX operating system in 1969.[1]
- UNIX was made at Bell Labs by Ken Thompson and Dennis Ritchie.
- UNIX introduced what is now called “the UNIX philosophy.”
- Almost all modern computing is derived from the legacy of UNIX.

[1] Dennis M. Ritchie and Ken Thompson. The UNIX time-sharing system. *Commun. ACM*, 17(7):365–375, jul 1974

UNIX

- The shell (as we recognize it) began with the UNIX operating system in 1969.[1]
- UNIX was made at Bell Labs by Ken Thompson and Dennis Ritchie.
- UNIX introduced what is now called “the UNIX philosophy.”
- Almost all modern computing is derived from the legacy of UNIX.

[1] Dennis M. Ritchie and Ken Thompson. The UNIX time-sharing system. *Commun. ACM*, 17(7):365–375, jul 1974

Outline

Anatomy of a Computer

Anatomy of a Computer

Input Keyboards, Mice, Serial Ports, etc.

Anatomy of a Computer

Input Keyboards, Mice, Serial Ports, etc.

Output Screens, Serial Ports, Speakers, etc.

Anatomy of a Computer

Input Keyboards, Mice, Serial Ports, etc.

Output Screens, Serial Ports, Speakers, etc.

Storage Memory (RAM), Disks, Disc Readers, etc.

Anatomy of a Computer

Input	Keyboards, Mice, Serial Ports, etc.
Output	Screens, Serial Ports, Speakers, etc.
Storage	Memory (RAM), Disks, Disc Readers, etc.
Compute	CPUs (math), FPUs (math with decimals), GPUs (math with matrices)

Anatomy of a Computer

Input	Keyboards, Mice, Serial Ports, etc.
Output	Screens, Serial Ports, Speakers, etc.
Storage	Memory (RAM), Disks, Disc Readers, etc.
Compute	CPUs (math), FPUs (math with decimals), GPUs (math with matrices)
Networking	Ethernet, Wi-Fi, Serial Ports, etc.

Anatomy of a Computer

Input	Keyboards, Mice, Serial Ports, etc.
Output	Screens, Serial Ports, Speakers, etc.
Storage	Memory (RAM), Disks, Disc Readers, etc.
Compute	CPUs (math), FPUs (math with decimals), GPUs (math with matrices)
Networking	Ethernet, Wi-Fi, Serial Ports, etc.
Misc.	Fans, Power Supplies, Sensors, etc.

Anatomy of a Computer

Input	Keyboards, Mice, Serial Ports, etc.
Output	Screens, Serial Ports, Speakers, etc.
Storage	Memory (RAM), Disks, Disc Readers, etc.
Compute	CPUs (math), FPUs (math with decimals), GPUs (math with matrices)
Networking	Ethernet, Wi-Fi, Serial Ports, etc.
Misc.	Fans, Power Supplies, Sensors, etc.

Anatomy of a Computer

Input	Keyboards, Mice, Serial Ports, etc.
Output	Screens, Serial Ports, Speakers, etc.
Storage	Memory (RAM), Disks, Disc Readers, etc.
Compute	CPUs (math), FPUs (math with decimals), GPUs (math with matrices)
Networking	Ethernet, Wi-Fi, Serial Ports, etc.
Misc.	Fans, Power Supplies, Sensors, etc.

Definition (kernel)

An Operating System Kernel is a program that abstracts over different hardware, allowing the same software to run on different computers.

Userspace

- A kernel by itself is kind of useless.

Userspace

- A kernel by itself is kind of useless.
- Abstractions are great, but we want to *do* something with our computers.

Userspace

- A kernel by itself is kind of useless.
- Abstractions are great, but we want to *do* something with our computers.
- This is where USERSPACE comes in.

Userspace

- A kernel by itself is kind of useless.
- Abstractions are great, but we want to *do* something with our computers.
- This is where USERSPACE comes in.

Definition (userspace)

USERSPACE is the set of programs that come bundled with an OS kernel, which allow a user to perform various tasks.

Running Programs

- Now that we have a bunch of programs installed, we want to run them.

Running Programs

- Now that we have a bunch of programs installed, we want to run them.
- We need something that wraps up all the programs and provides a common interface to them.

Running Programs

- Now that we have a bunch of programs installed, we want to run them.
- We need something that wraps up all the programs and provides a common interface to them.

Definition (shell)

A SHELL is the outermost layer of an operating system; it lets a user run userspace programs, which in turn let a user interact with their computer's hardware.

Running Programs

- Now that we have a bunch of programs installed, we want to run them.
- We need something that wraps up all the programs and provides a common interface to them.

Definition (shell)

A SHELL is the outermost layer of an operating system; it lets a user run userspace programs, which in turn let a user interact with their computer's hardware.

Definition (operating system)

An OPERATING SYSTEM is the combination of a kernel, a set of userspace programs, and a shell.

Types of Shell

Operating System	Shell	Type	How you start programs
Windows	`explorer.exe`	Graphical	Start Menu, Desktop
macOS	Aqua	Graphical	Dock, Launchpad
iOS, Android	Home Screen	Graphical	Tap icon
Linux	GNOME, KDE, XFCE, ...	Graphical	Various

Types of Shell

Operating System	Shell	Type	How you start programs
Windows	`explorer.exe`	Graphical	Start Menu, Desktop
macOS	Aqua	Graphical	Dock, Launchpad
iOS, Android	Home Screen	Graphical	Tap icon
Linux	GNOME, KDE, XFCE, ...	Graphical	Various
Windows	`cmd.exe`	Text	Type name of `.exe` file
UNIX	`sh`	Text	The rest of this lecture.
Linux	`bash`	Text	Same as `sh`
macOS	`zsh`	Text	Same as `sh`

Table: Shells across common operating systems

Types of Shell

Operating System	Shell	Type	How you start programs
Windows	`explorer.exe`	Graphical	Start Menu, Desktop
macOS	Aqua	Graphical	Dock, Launchpad
iOS, Android	Home Screen	Graphical	Tap icon
Linux	GNOME, KDE, XFCE, ...	Graphical	Various
Windows	`cmd.exe`	Text	Type name of `.exe` file
UNIX	`sh`	Text	The rest of this lecture.
Linux	`bash`	Text	Same as `sh`
macOS	`zsh`	Text	Same as `sh`

Table: Shells across common operating systems

While all of these shells can *start* programs, only the UNIX shell (and its derivatives) can *combine* them.

Types of Shell

Operating System	Shell	Type	How you start programs
Windows	`explorer.exe`	Graphical	Start Menu, Desktop
macOS	Aqua	Graphical	Dock, Launchpad
iOS, Android	Home Screen	Graphical	Tap icon
Linux	GNOME, KDE, XFCE, ...	Graphical	Various
Windows	`cmd.exe`	Text	Type name of `.exe` file
UNIX	`sh`	Text	The rest of this lecture.
Linux	`bash`	Text	Same as `sh`
macOS	`zsh`	Text	Same as `sh`

Table: Shells across common operating systems

While all of these shells can *start* programs, only the UNIX shell (and its derivatives) can *combine* them.

Outline

Original

As described in the Bell System Technical Journal in 1978 [2]:

1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features."
2. Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.
3. Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.
4. Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.

---

[2] M. D. McIlroy, E. N. Pinson, and B. A. Tague. UNIX time-sharing system: Foreword. *The Bell System Technical Journal*, 57(6):1899–1904, July 1978

Simplified

The UNIX Philosophy

Build lots of small tools, each of which does exactly one thing well, but which can be combined to do more powerful things.

Outline

The UNIX File Abstraction

- In UNIX, everything is a file (including hardware!).

The UNIX File Abstraction

- In UNIX, everything is a file (including hardware!).
- Most files are text.

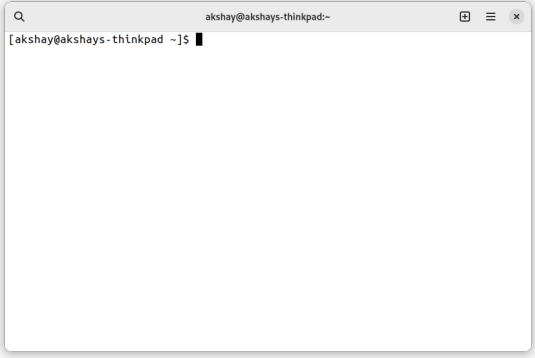
The UNIX File Abstraction

- In UNIX, everything is a file (including hardware!).
- Most files are text.
- Programs which operate on text can operate on almost everything.

Outline

1. What is the Shell?
2. The UNIX Shell
3. Basic Commands
4. Pipes
5. Conclusion

The UNIX Shell



The Prompt

Example (prompt)

An default shell prompt might look like this:

```
[akshay@akshays-thinkpad ~]$
```

This PROMPT will print every time the shell is ready to accept another command. It probably looks different on your computer, since different shells have different defaults. Regardless of what it looks like right now, you can customize it to look like whatever you want. In most cases, you can get back to it by pressing CTRL-C on your keyboard.

The Prompt

Username

Example (prompt: username)

This part of the prompt is your username:

```
[akshay@akshays-thinkpad ~]$
```

This is probably the same as the username you use to log into your computer. By default, you're auto-logged into the shell using your normal user account.

The Prompt

Hostname

Example (prompt: hostname)

This part of the prompt is your computer's hostname:

```
[akshay@akshays-thinkpad ~]$
```

This is your computer's name on whatever network it's connected to. Generally you don't really care about this unless you have multiple computers.

The Prompt

Current Directory

Example (prompt: current directory)

This part of the prompt is your current directory.

```
[akshay@akshays-thinkpad ~]$
```

This is your WORKING DIRECTORY; "directory" is just a fancy name for "folder", like you'd have in Windows Explorer or macOS Finder.

By default, you start in your HOME DIRECTORY, which is the folder that contains `Documents`, `Downloads`, `Pictures`, `Videos`, etc. The home directory is abbreviated as a tilde (~) because it's so common.

Outline

Outline

Listing Files

In Windows Explorer or macOS Finder, the current directory is always visible. In the shell, you have to ask for the list of current files manually.

Listing Files

In Windows Explorer or macOS Finder, the current directory is always visible. In the shell, you have to ask for the list of current files manually.

The LIST command is called `ls`.

Listing Files

In Windows Explorer or macOS Finder, the current directory is always visible. In the shell, you have to ask for the list of current files manually.

The LIST command is called `ls`.

Example (ls)

On my computer, the results look like this:

```
[akshay@akshays-thinkpad ~]$ ls
Desktop    Downloads  Music     Public     Videos
Documents  Dropbox    Pictures  Templates
[akshay@akshays-thinkpad ~]$
```

These are all the SUBDIRECTORIES of my home directory.

You are here

- Before we go exploring, let's do one more command in the home directory.

You are here

- Before we go exploring, let's do one more command in the home directory.

The PRINT WORKING DIRECTORY command is called `pwd`.

You are here

- Before we go exploring, let's do one more command in the home directory.

The PRINT WORKING DIRECTORY command is called `pwd`.

Example (pwd)

Print the current working directory:

```
[akshay@akshays-thinkpad ~]$ pwd
/home/akshay
[akshay@akshays-thinkpad ~]$
```

You are here

- Before we go exploring, let's do one more command in the home directory.

The PRINT WORKING DIRECTORY command is called `pwd`.

Example (pwd)

Print the current working directory:

```
[akshay@akshays-thinkpad ~]$ pwd
/home/akshay
[akshay@akshays-thinkpad ~]$
```

Definition (root directory)

The ROOT DIRECTORY is the topmost directory on the filesystem. It's often called `/`.

Changing Directories

- The home directory isn't too interesting on its own.
- Let's go somewhere you probably know well: the Desktop!
- In Explorer or Finder you could just click on a folder to enter it.

Changing Directories

- The home directory isn't too interesting on its own.
- Let's go somewhere you probably know well: the Desktop!
- In Explorer or Finder you could just click on a folder to enter it.

The CHANGE DIRECTORY command is called `cd`.

Changing Directories

- The home directory isn't too interesting on its own.
- Let's go somewhere you probably know well: the Desktop!
- In Explorer or Finder you could just click on a folder to enter it.

The CHANGE DIRECTORY command is called `cd`.

Example (cd)

Change (`cd`) into the `Desktop` directory:

```
[akshay@akshays-thinkpad ~]$ cd Desktop
[akshay@akshays-thinkpad Desktop]$
```

Not Changing Directories

- There's a special name that always means “the current directory”: `.`
- `cd .` says “change directory to the current directory”.

Not Changing Directories

- There's a special name that always means “the current directory”: `.`
- `cd .` says “change directory to the current directory”.

Example (cd .)

Don’t change directories:

```
[akshay@akshays-thinkpad Desktop]$ pwd
/home/akshay/Desktop
[akshay@akshays-thinkpad Desktop]$ cd .
[akshay@akshays-thinkpad Desktop]$ pwd
/home/akshay/Desktop
```

Making Directories

- Now that we're on the desktop, let's create a new directory to do some experiments in.
- This is the equivalent of right-clicking and selecting “New Folder”.

Making Directories

- Now that we're on the desktop, let's create a new directory to do some experiments in.
- This is the equivalent of right-clicking and selecting “New Folder”.

The MAKE DIRECTORY command is called `mkdir`.

Making Directories

- Now that we're on the desktop, let's create a new directory to do some experiments in.
- This is the equivalent of right-clicking and selecting “New Folder”.

The MAKE DIRECTORY command is called `mkdir`.

Example (mkdir)

Create a directory called “cs45-test-directory” and `cd` into it:

```
$ mkdir cs45-test-directory
$ cd cs45-test-directory/
```

You can go back to the Desktop by typing `cd ..` or `cd ~/Desktop`.

Making Directories

- Now that we're on the desktop, let's create a new directory to do some experiments in.
- This is the equivalent of right-clicking and selecting “New Folder”.

The MAKE DIRECTORY command is called `mkdir`.

Example (mkdir)

Create a directory called “cs45-test-directory” and `cd` into it:

```
$ mkdir cs45-test-directory
$ cd cs45-test-directory/
```

You can go back to the Desktop by typing `cd ..` or `cd ~/Desktop`.

The REMOVE DIRECTORY command is `rmdir`.

Directory Review

Shortcut	Name
~	Home Directory
/	Root Directory
.	Current Directory
. .	Parent Directory

Table: Directory Shortcuts

Directory Review

Shortcut	Name
~	Home Directory
/	Root Directory
.	Current Directory
. .	Parent Directory

Table: Directory Shortcuts

Command	Description	Argument	Required
`ls`	List Directory	Directory Name	No, defaults to .
`cd`	Change Directory	Directory Name	No, defaults to ~
`pwd`	Print Working Directory	N/A	N/A
`mkdir`	Make Directory	Directory Name	Yes
`rmdir`	Remove Directory	Directory Name	Yes

Table: Directory Commands

Outline

Output

- Sometimes we just want to print something out, like “hello, world”.

Output

- Sometimes we just want to print something out, like “hello, world”.

The PRINT command is called `echo`.

Output

- Sometimes we just want to print something out, like “hello, world”.

The PRINT command is called `echo`.

Example (echo)

Print the text “hello, world”:

```
$ echo "hello, world"
hello, world
$
```

Input

- Sometimes we want to read input from the user.

Input

- Sometimes we want to read input from the user.
- Unfortunately, this is trickier: there are multiple commands which can be used for input. Let's use the simplest one.

Input

- Sometimes we want to read input from the user.
- Unfortunately, this is trickier: there are multiple commands which can be used for input. Let's use the simplest one.

The CONCATENATE command is called `cat`. It can also be used for input.

Input

The CONCATENATE command is called `cat`. It can also be used for input.

Example (cat)

Read text from the user:

```
$ cat
this is a test
this is a test
line 2
line 2
$
```

The `cat` command will print out whatever you type into it... forever. To get it to stop, you can either "kill" it by pressing CTRL-C, or tell it "end of file" by pressing CTRL-D.

Creating Files

- There are a few different ways to create files. Let's start with the simplest.

Creating Files

- There are a few different ways to create files. Let's start with the simplest.

The TOUCH FILE command is called, intuitively, `touch`. While we don't care about "touching" files as such, it has the handy side effect of creating files.

Creating Files

- There are a few different ways to create files. Let's start with the simplest.

The TOUCH FILE command is called, intuitively, `touch`. While we don't care about "touching" files as such, it has the handy side effect of creating files.

Example (touch)

To create a file called "text.txt":

```
$ touch test.txt
$ ls
test.txt
$
```

Renaming Files

The MOVE FILE command is called `mv`. It can also be used to rename files.

Renaming Files

The MOVE FILE command is called `mv`. It can also be used to rename files.

Example (mv)

To rename a file called “text.txt” to “empty.txt”:

```
$ mv test.txt empty.txt
$ ls
empty.txt
$
```

Deleting Files

The REMOVE FILE command is called `rm`.

This is irreversible!

This command is dangerous! It does **not** move the file to a “trash” folder; it permanently and irreversibly deletes it.

Deleting Files

The REMOVE FILE command is called `rm`.

This is irreversible!

This command is dangerous! It does **not** move the file to a "trash" folder; it permanently and irreversibly deletes it.

Example (rm)

To remove a file called "text.txt":

```
$ rm test.txt
$
```

Writing to Files

- We have a problem though: the file we created is empty. We can't do much with a bunch of empty files.
- We can check this by running `ls` with a special `FLAG` asking for extra info (including the file size).

Writing to Files

- We have a problem though: the file we created is empty. We can't do much with a bunch of empty files.
- We can check this by running `ls` with a special `FLAG` asking for extra info (including the file size).

Example (ls -l)

To print extra information about files:

```
$ ls -l
total 0
-rw-r--r-- 1 akshay akshay 0 Dec 18 11:59 test.txt
```

Writing to Files

- We have a problem though: the file we created is empty. We can't do much with a bunch of empty files.
- We can check this by running `ls` with a special `FLAG` asking for extra info (including the file size).

Example (ls -l)

To print extra information about files:

```
$ ls -l
total 0
-rw-r--r-- 1 akshay akshay 0 Dec 18 11:59 test.txt
```

One very useful flag which is supported by almost every command is `--help`.

Writing to Files

- Everything is a file, including the output of our commands.

Writing to Files

- Everything is a file, including the output of our commands.
- By default, this is called STANDARD OUTPUT, and goes to our terminal.

Writing to Files

- Everything is a file, including the output of our commands.
- By default, this is called STANDARD OUTPUT, and goes to our terminal.
- The shell lets us REDIRECT standard output to go to a file instead.

Writing to Files

- Everything is a file, including the output of our commands.
- By default, this is called STANDARD OUTPUT, and goes to our terminal.
- The shell lets us REDIRECT standard output to go to a file instead.

Example (output redirection)

To create a file called "hello.txt" with the contents `hello, world`:

```
$ echo "hello, world" > hello.txt
$
```

To append to an existing file, you can use `>>` instead of `>`.

Reading from Files

- Just like STANDARD OUTPUT, the input to our programs is also a file.

Reading from Files

- Just like STANDARD OUTPUT, the input to our programs is also a file.
- By default, this is called STANDARD INPUT, and comes from our terminal.

Reading from Files

- Just like STANDARD OUTPUT, the input to our programs is also a file.
- By default, this is called STANDARD INPUT, and comes from our terminal.
- The shell also lets us REDIRECT standard input to come from a file.

Reading from Files

- Just like STANDARD OUTPUT, the input to our programs is also a file.
- By default, this is called STANDARD INPUT, and comes from our terminal.
- The shell also lets us REDIRECT standard input to come from a file.

Example (input redirection)

To print a file called "hello.txt":

```
$ cat < hello.txt
hello, world
$
```

I/O(/E?)

Definition (standard input)

STANDARD INPUT (`/dev/stdin`) is the file from which a program reads its input.

I/O(/E?)

Definition (standard input)

STANDARD INPUT (`/dev/stdin`) is the file from which a program reads its input.

Definition (standard output)

STANDARD OUTPUT (`/dev/stdout`) is the file to which a program writes its output.

I/O(/E?)

Definition (standard input)

STANDARD INPUT (`/dev/stdin`) is the file from which a program reads its input.

Definition (standard output)

STANDARD OUTPUT (`/dev/stdout`) is the file to which a program writes its output.

Definition (standard error)

STANDARD ERROR (`/dev/stderr`) is the file to which a program writes its error messages.

Redirection Operators

Operator	File	Overwrite?
<	/dev/stdin	
>	/dev/stdout	Overwrite
>>	/dev/stdout	Append
2>	/dev/stderr[3]	Overwrite
2>>	/dev/stderr	Append

Table: UNIX Shell Redirection Operators

[3] It's uncommon to redirect standard error, but there are some valid reasons to (which we'll see later in the quarter).

Outline

1. What is the Shell?
2. The UNIX Shell
3. Basic Commands
4. Pipes
5. Conclusion

Environment Variables

Some programs need configuration that's too annoying to provide as arguments every time.

Environment Variables

Some programs need configuration that's too annoying to provide as arguments every time.

Definition (environment variable)

An ENVIRONMENT VARIABLE is a configuration value that's set globally by a program, which applies to itself and any other programs it runs.

All the Environment Variables

The ENVIRONMENT VARIABLE command `env` prints all the environment variables which are currently set.

All the Environment Variables

The ENVIRONMENT VARIABLE command `env` prints all the environment variables which are currently set.

Example (env)

To print every environment variable:

```
$ env
MAIL=/var/spool/mail/akshay
PWD=/home/akshay
XDG_SESSION_TYPE=wayland
PATH=/usr/local/bin:/usr/bin:/usr/local/sbin
HOME=/home/akshay
USERNAME=akshay
[...]
```

Pipes

- Let's see how many environment variables we have!

Pipes

- Let's see how many environment variables we have!
- The WORD COUNT command is called `wc`.

Pipes

- Let's see how many environment variables we have!
- The WORD COUNT command is called `wc`.
- It has a flag `--lines` (or `-l`) which counts lines in its input instead of words.

Pipes

- Let's see how many environment variables we have!
- The WORD COUNT command is called `wc`.
- It has a flag `--lines` (or `-l`) which counts lines in its input instead of words.

Example (count environment variables with a temporary file)

We can write the output into a temporary file, and give it as input to `wc`:

```
$ env > /tmp/env.txt
$ wc --lines < /tmp/env.txt
78
```

Pipes

- Let's see how many environment variables we have!
- The WORD COUNT command is called `wc`.
- It has a flag `--lines` (or `-l`) which counts lines in its input instead of words.

Example (count environment variables with a pipe)

We can connect the output of `env` and the input of `wc` with a PIPE:

```
$ env | wc --lines
78
```

Benefits of Pipes

Definition (pipe)

A PIPE is a direct connection between the output of one program and the input of another. It can be set up using the | (pipe) operator, which connects `stdout` of whatever is on the left with `stdin` of whatever is on the right.

Benefits of Pipes

Definition (pipe)

A PIPE is a direct connection between the output of one program and the input of another. It can be set up using the | (pipe) operator, which connects `stdout` of whatever is on the left with `stdin` of whatever is on the right.

Pipes are superior to temporary files for several reasons:

Benefits of Pipes

Definition (pipe)

A PIPE is a direct connection between the output of one program and the input of another. It can be set up using the | (pipe) operator, which connects `stdout` of whatever is on the left with `stdin` of whatever is on the right.

Pipes are superior to temporary files for several reasons:

- They are **parallel**: the programs on the left and right can run at the same time.

Benefits of Pipes

Definition (pipe)

A PIPE is a direct connection between the output of one program and the input of another. It can be set up using the | (pipe) operator, which connects `stdout` of whatever is on the left with `stdin` of whatever is on the right.

Pipes are superior to temporary files for several reasons:

- They are **parallel**: the programs on the left and right can run at the same time.
- They are **lazy**: the program on the right can read exactly as much data as it needs from the program on the left.

More Piping

Example (the first *n* environment variables)

With the `head` command, we can extract only the first few lines from a file:

```
$ env | head --lines=3
MAIL=/var/spool/mail/akshay
PWD=/home/akshay
XDG_SESSION_TYPE=wayland
$
```

More Piping

Example (random numbers)

We can lazily evaluate part of an infinitely long "file" such as `/dev/random`:

```
$ cat /dev/random | hexdump | head --lines 1
0000000 4730 003c 6c22 1d16 49ef 6eff 91b2 a9f0
```

Outline

1. What is the Shell?
2. The UNIX Shell
3. Basic Commands
4. Pipes
5. Conclusion

Getting Help

- The shell is far more complicated than we can possibly cover in an 80-minute lecture.

Getting Help

- The shell is far more complicated than we can possibly cover in an 80-minute lecture.
- Most commands have lots of flags and options.

Getting Help

- The shell is far more complicated than we can possibly cover in an 80-minute lecture.
- Most commands have lots of flags and options.
- We already talked about the `--help` flag, which usually gives you a brief summary of how to use a command.

The System Manual

- One super-useful resource is the UNIX system manual, which is pre-installed on most UNIX-like systems.

The System Manual

- One super-useful resource is the UNIX system manual, which is pre-installed on most UNIX-like systems.
- The MANUAL command is `man`; it takes as an argument the name of a command, and it displays the manual page ("man page").

The System Manual

- One super-useful resource is the UNIX system manual, which is pre-installed on most UNIX-like systems.
- The MANUAL command is `man`; it takes as an argument the name of a command, and it displays the manual page ("man page").
- If you don't know the name of a command, you can search the manual using the command `apropos` (or, equivalently, `man --apropos`).

The System Manual

- One super-useful resource is the UNIX system manual, which is pre-installed on most UNIX-like systems.
- The MANUAL command is `man`; it takes as an argument the name of a command, and it displays the manual page ("man page").
- If you don't know the name of a command, you can search the manual using the command `apropos` (or, equivalently, `man --apropos`).

Example (man wc)

To open the UNIX manual page for the `wc` word-count tool:

```
$ man wc
```

The System Manual

- One super-useful resource is the UNIX system manual, which is pre-installed on most UNIX-like systems.
- The MANUAL command is `man`; it takes as an argument the name of a command, and it displays the manual page ("man page").
- If you don't know the name of a command, you can search the manual using the command `apropos` (or, equivalently, `man --apropos`).

Example (man man)

To open the UNIX manual page for the manual itself:

```
$ man man
```

To be continued...

- We'll continue exploring shell tools in Lecture 3: *Data Manipulation and Shell Scripting*.

To be continued...

- We'll continue exploring shell tools in Lecture 3: *Data Manipulation and Shell Scripting*.
- There's no class on Monday, January 16th (Martin Luther King Jr. Day); we'll pick back up on Wednesday, January 18th.

To be continued...

- We'll continue exploring shell tools in Lecture 3: *Data Manipulation and Shell Scripting*.
- There's no class on Monday, January 16th (Martin Luther King Jr. Day); we'll pick back up on Wednesday, January 18th.
- We'll post Assignment 1 soon, but it'll only be due on Monday, January 23rd (twelve days from now). It'll cover Lectures 2 (this one) and 3 (next week).

To be continued... (Continued)

- In the meantime: try doing file management from the terminal. We didn't cover every command you'll need, so if you don't know how to do something, try searching the manual using `apropos` or searching the web.

To be continued... (Continued)

- In the meantime: try doing file management from the terminal. We didn't cover every command you'll need, so if you don't know how to do something, try searching the manual using `apropos` or searching the web.
- Take a look at the UNIX filesystem hierarchy layout, documented in `man hier`.
- Use `man` pages, `http://cheat.sh/`, or `https://devhints.io/bash` to find out more about shell commands.

To be continued... (Continued)

- In the meantime: try doing file management from the terminal. We didn't cover every command you'll need, so if you don't know how to do something, try searching the manual using `apropos` or searching the web.
- Take a look at the UNIX filesystem hierarchy layout, documented in `man hier`.
- Use `man` pages, `http://cheat.sh/`, or `https://devhints.io/bash` to find out more about shell commands.

Be Careful!

The shell often doesn't warn you when you're doing dangerous things! Be sure to read the `man` page before running commands you find on the internet. Be especially careful with the REMOVE FILE command, `rm`, or when using the > (overwrite) operator.

Sudo

Using `sudo` responsibly

- Some commands require the use of `sudo`, the SUPERUSER DO command.

Sudo

Using `sudo` responsibly

- Some commands require the use of `sudo`, the SUPERUSER DO command.
- This gives that command full access to do anything on your computer!

Sudo

Using `sudo` responsibly

- Some commands require the use of `sudo`, the SUPERUSER DO command.
- This gives that command full access to do anything on your computer!
- Sometimes `sudo` won't even ask for your password!

Sudo

Using `sudo` responsibly

- Some commands require the use of `sudo`, the SUPERUSER DO command.
- This gives that command full access to do anything on your computer!
- Sometimes `sudo` won't even ask for your password!
- If you're using `sudo`, make sure you know what the command after it will do.

Sudo

Using `sudo` responsibly

- Some commands require the use of `sudo`, the SUPERUSER DO command.
- This gives that command full access to do anything on your computer!
- Sometimes `sudo` won't even ask for your password!
- If you're using `sudo`, make sure you know what the command after it will do.
- `sudo` is necessary for certain tasks (we'll see some in the next few lectures), but it's always good to be careful around it.

Sudo Warning

```
We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.
```

Interesting Commands

`head`: Get the beginning of a file (or pipe).

`tail`: Get the end of a file (or pipe).

`grep`: Search within a file.

`sed`: Find-and-replace.

`cut`: Get a specific “column” of a file (e.g., a CSV file).

`ping`: Test your internet connection.

`sort`: Sort lines in a file.

`uniq`: Remove duplicate lines in a file.

`exit`: Exit the terminal.

Questions?

References

[1] M. D. McIlroy, E. N. Pinson, and B. A. Tague. UNIX time-sharing system: Foreword. *The Bell System Technical Journal*, 57(6):1899—1904, July 1978.

[2] Dennis M. Ritchie and Ken Thompson. The UNIX time-sharing system. *Commun. ACM*, 17(7):365–375, jul 1974.