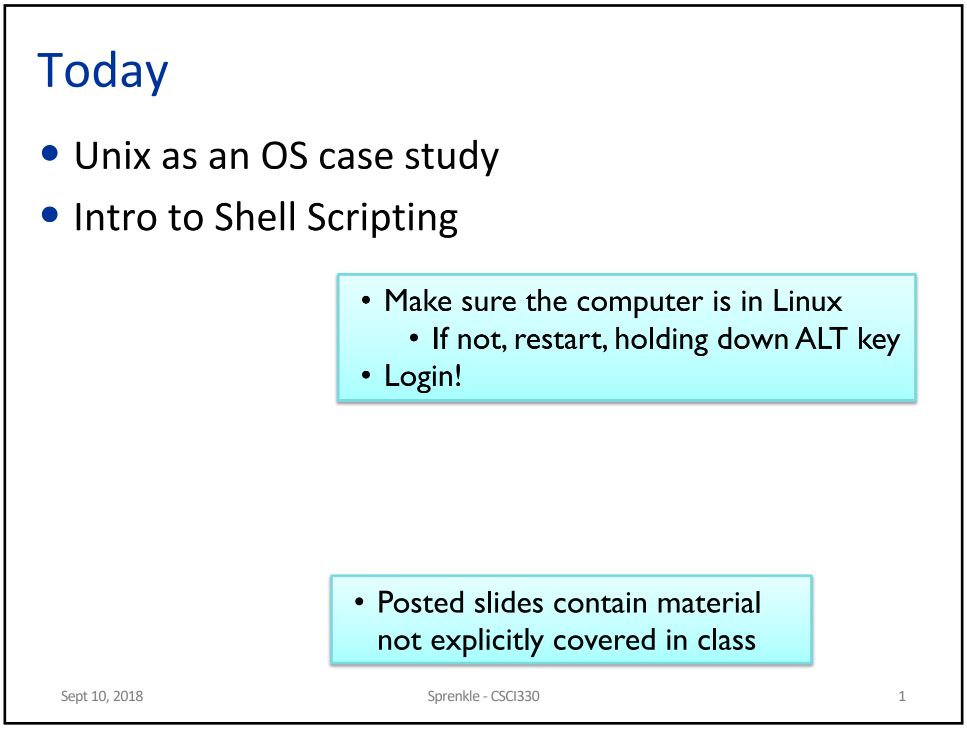
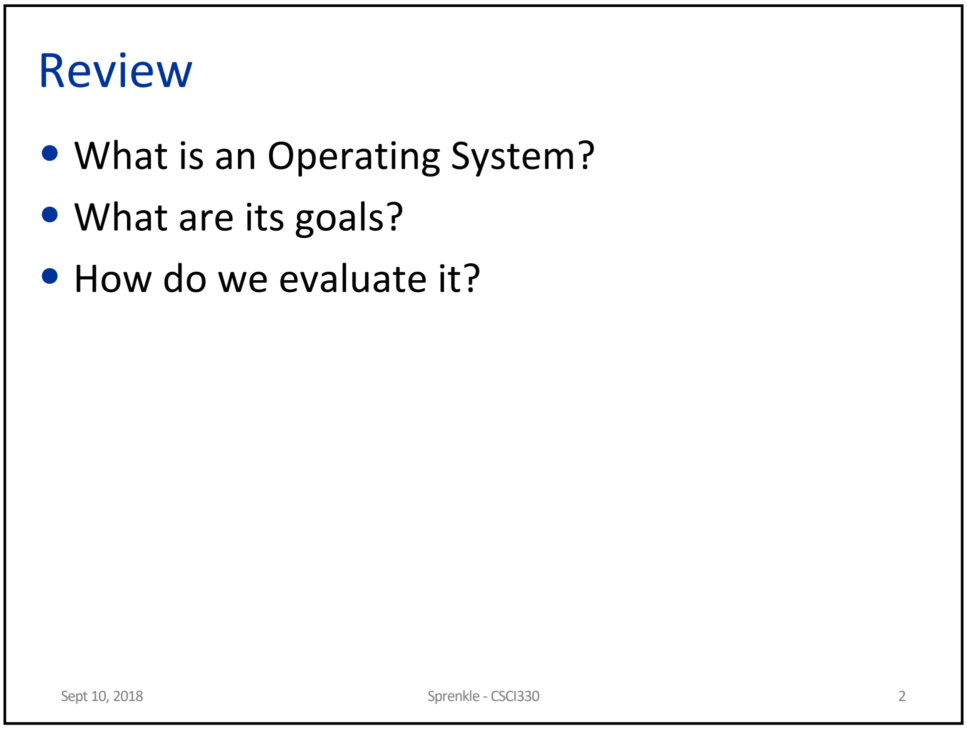


Today

- Unix as an OS case study
- Intro to Shell Scripting

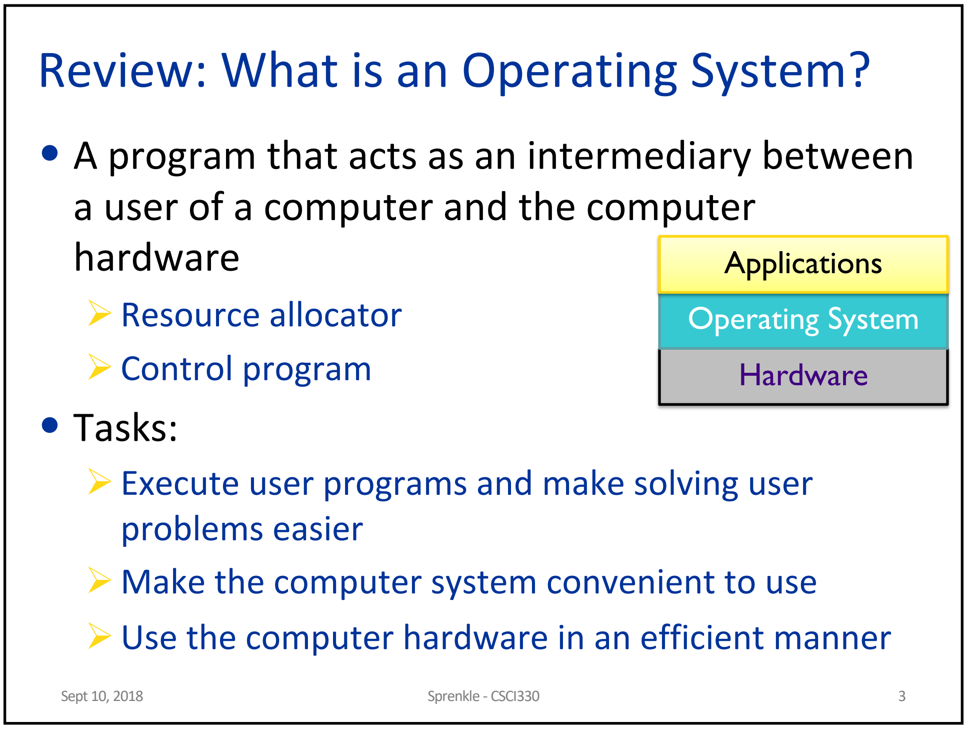
- Make sure the computer is in Linux
  - If not, restart, holding down ALT key
- Login!

- Posted slides contain material not explicitly covered in class



Review

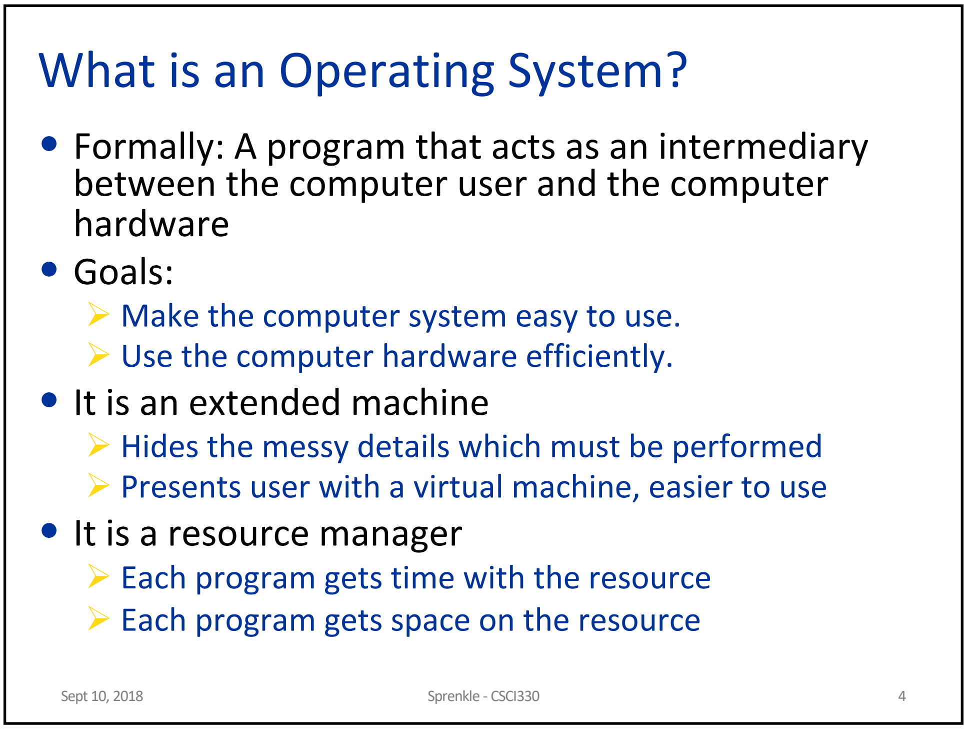
- What is an Operating System?
- What are its goals?
- How do we evaluate it?



Review: What is an Operating System?

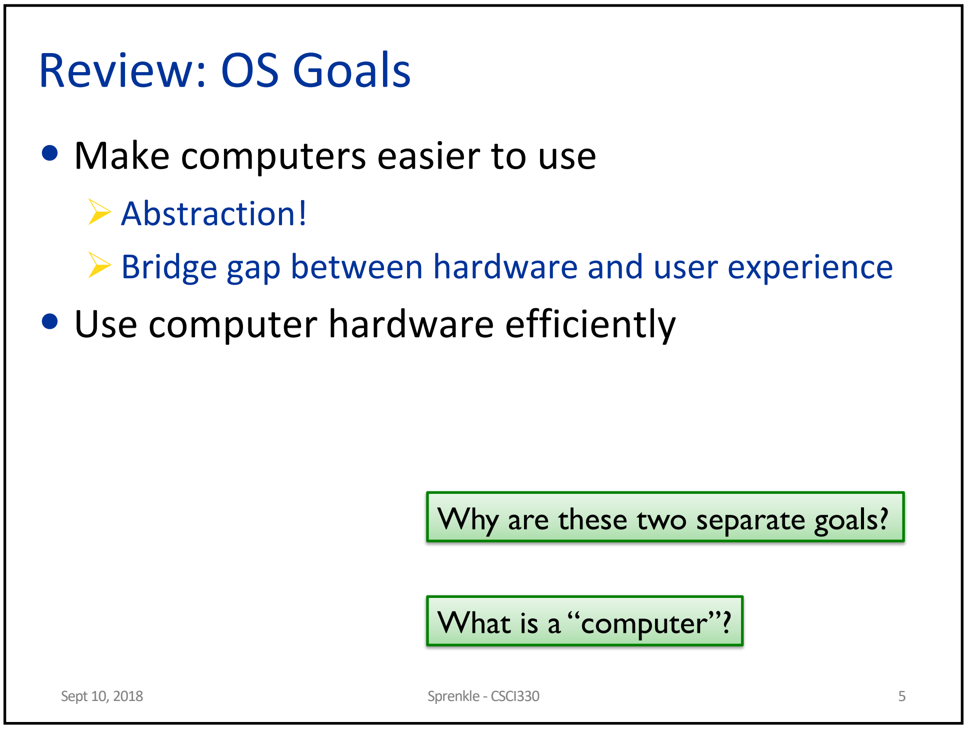
- A program that acts as an intermediary between a user of a computer and the computer hardware
  - ➢ Resource allocator
  - ➢ Control program
- Tasks:
  - ➢ Execute user programs and make solving user problems easier
  - ➢ Make the computer system convenient to use
  - ➢ Use the computer hardware in an efficient manner

Applications
Operating System
Hardware



What is an Operating System?

- Formally: A program that acts as an intermediary between the computer user and the computer hardware
- Goals:
  - ➢ Make the computer system easy to use.
  - ➢ Use the computer hardware efficiently.
- It is an extended machine
  - ➢ Hides the messy details which must be performed
  - ➢ Presents user with a virtual machine, easier to use
- It is a resource manager
  - ➢ Each program gets time with the resource
  - ➢ Each program gets space on the resource

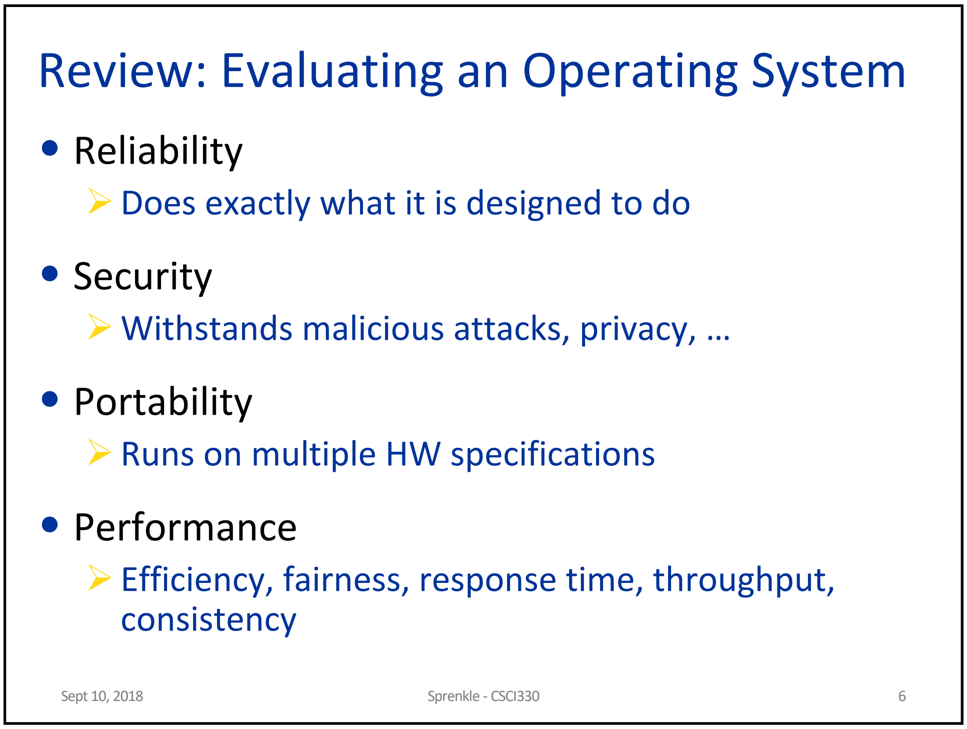


Review: OS Goals

- Make computers easier to use
  - Abstraction!
  - Bridge gap between hardware and user experience
- Use computer hardware efficiently

Why are these two separate goals?

What is a “computer”?



Review: Evaluating an Operating System

- Reliability
  - Does exactly what it is designed to do
- Security
  - Withstands malicious attacks, privacy, …
- Portability
  - Runs on multiple HW specifications
- Performance
  - Efficiency, fairness, response time, throughput, consistency

SYSTEMS PROGRAMMING

One Course Goal: Develop a Simple OS

- How are we going to do that?
  - Systems programming!

What is Systems Programming?

- Program development with system tools
  - (no fancy pants IDEs here)
- Uses system calls that hook in to core OS functions
- Use coding standards to ensure portability
  - Common file locations
  - Common compilation & installation procedures
  - Basic shell functionality
- We’ll be programming in the Unix environment, using C

The System Programmer’s Toolbox

- Shell: a program used to run other programs
- Text editor: where you’ll develop your code
  - Your faves?
- Compiler: transforms source code into an executable file
  - gcc
- Debugger: a program that allows you to step through an execution & observe how the program state (i.e., variable values) changes
  - gdb
  - Print statements

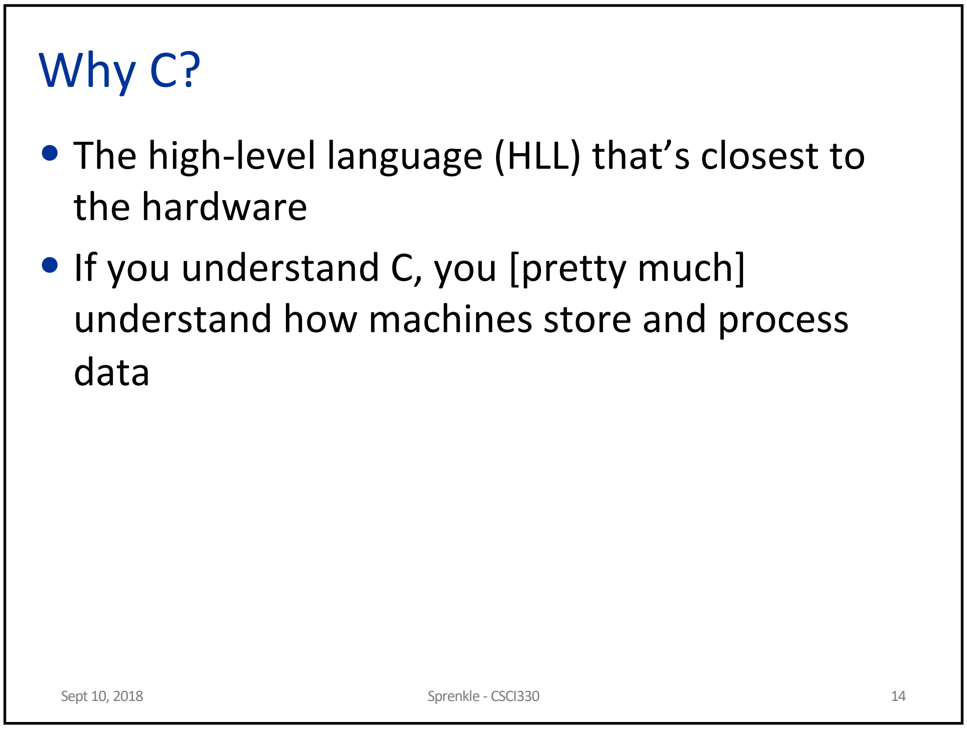
The System Programmer’s Toolbox

- Shell: a program used to run other programs
- Text editor: where you’ll develop your code
  - Your faves?
- Compiler: transforms source code into an executable file
  - gcc
- Debugger: a program that allows you to step through an execution & observe how the program state (i.e., variable values) changes
  - gdb
  - Print statements

More on Wednesday

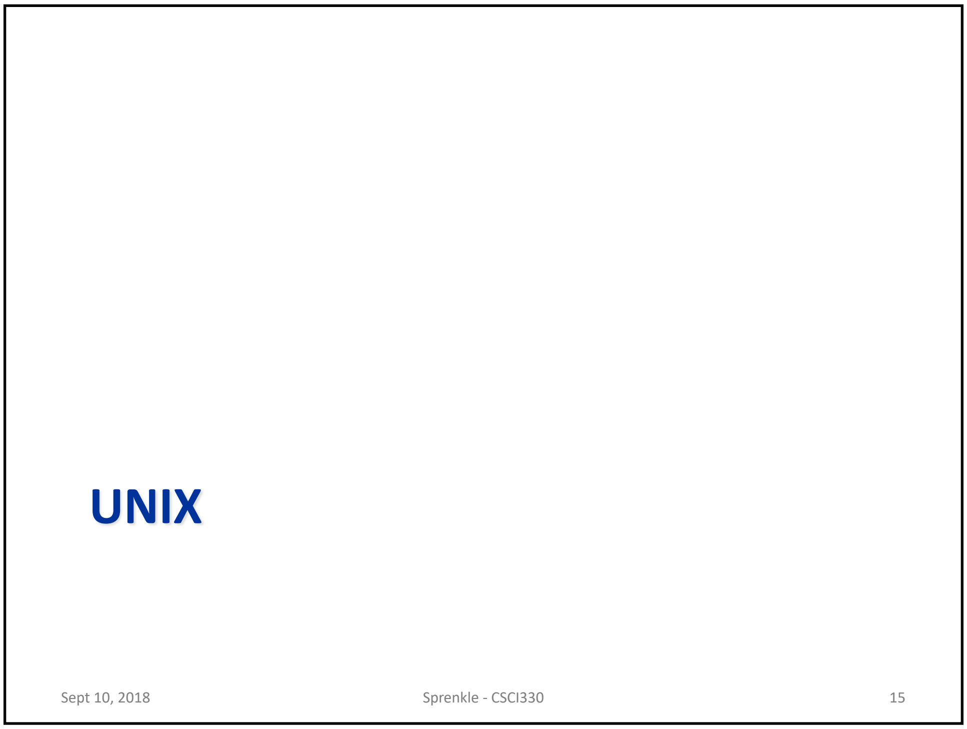
Why Unix?

- Open source = easier to study
  - Windows is proprietary & closed
  - OSX is proprietary and is built on top of Unix
- Historic: developed in the 60s & 70s
  - One of the oldest OS’s in use today
- Most serious programmers and hackers know their way around Unix/Linux
- Linux is a Unix-like OS



Why C?

- The high-level language (HLL) that's closest to the hardware
- If you understand C, you [pretty much] understand how machines store and process data



UNIX

Unix Philosophy

- Make each program do one thing well
  - More complex functionality by combining programs
  - Make every program a filter
  - More efficient
  - Better for reuse
- Portability
- No GUIs
- Only error feedback

What is a Shell?

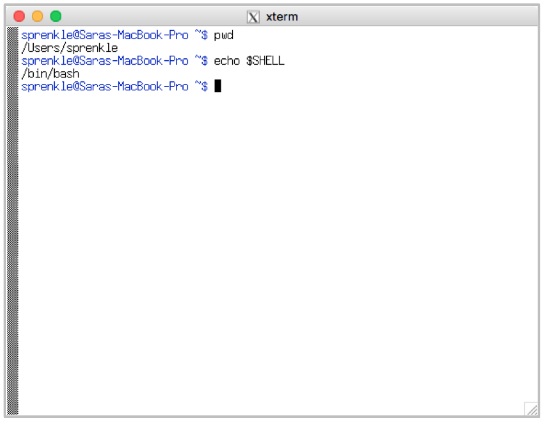
- User interface to the operating system
- Command-line ***interpreter***
- Functionality:
  - Execute other programs
  - Manage files
  - Manage processes
- A program, like any other
- Basic form of shell:

```
while <read command>:
    parse command
    execute command
```

hides details of underlying operating system

The Shell and Terminal

- When you open the terminal, you can interact with the shell

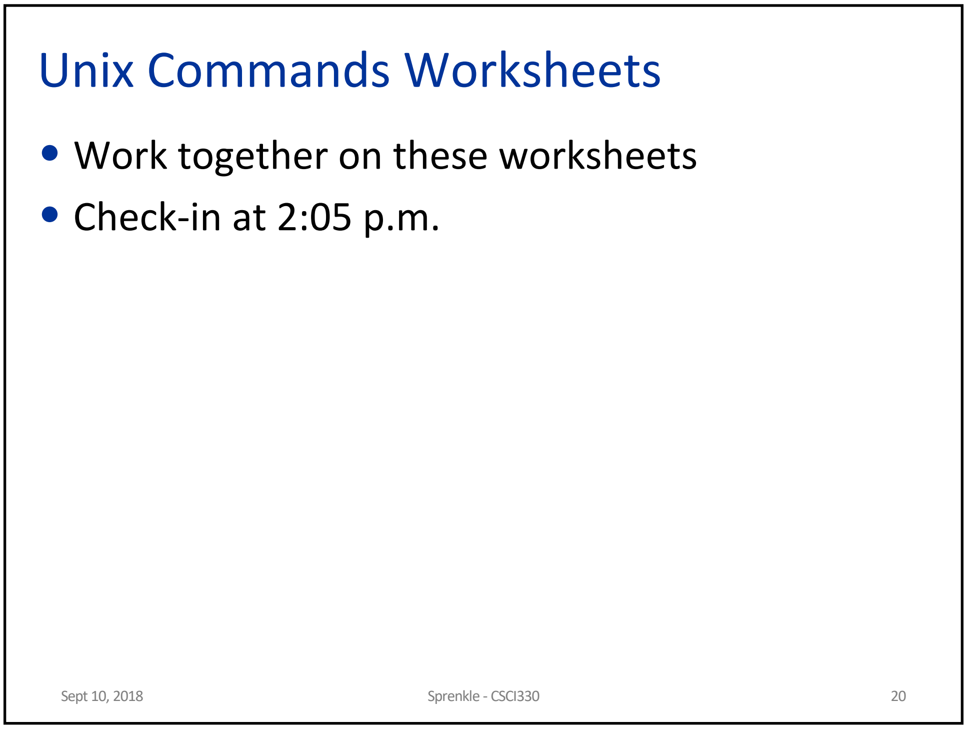


```
sprenkle@Saras-MacBook-Pro ~$ pwd
/Users/sprenkle
sprenkle@Saras-MacBook-Pro ~$ echo $SHELL
/bin/bash
sprenkle@Saras-MacBook-Pro ~$
```

Directory Shortcuts

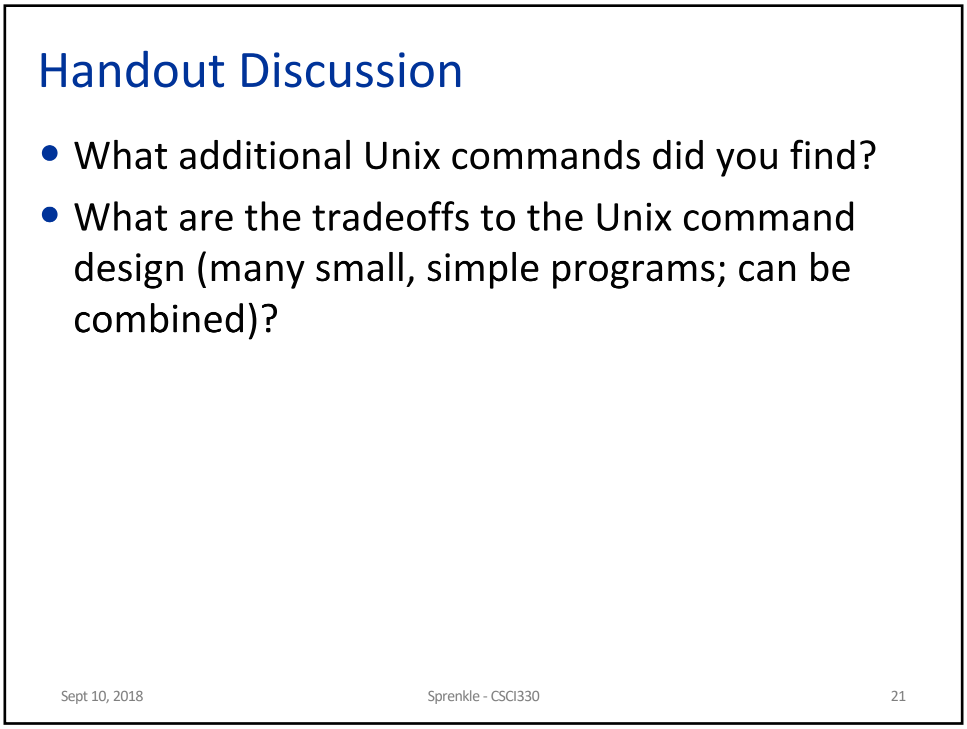
- .
  - Current directory
- . .
  - Parent directory of current directory
  - Every directory except the root directory has a parent directory
- ~
  - User’s home directory

Useful in a variety of Unix commands



Unix Commands Worksheets

- Work together on these worksheets
- Check-in at 2:05 p.m.



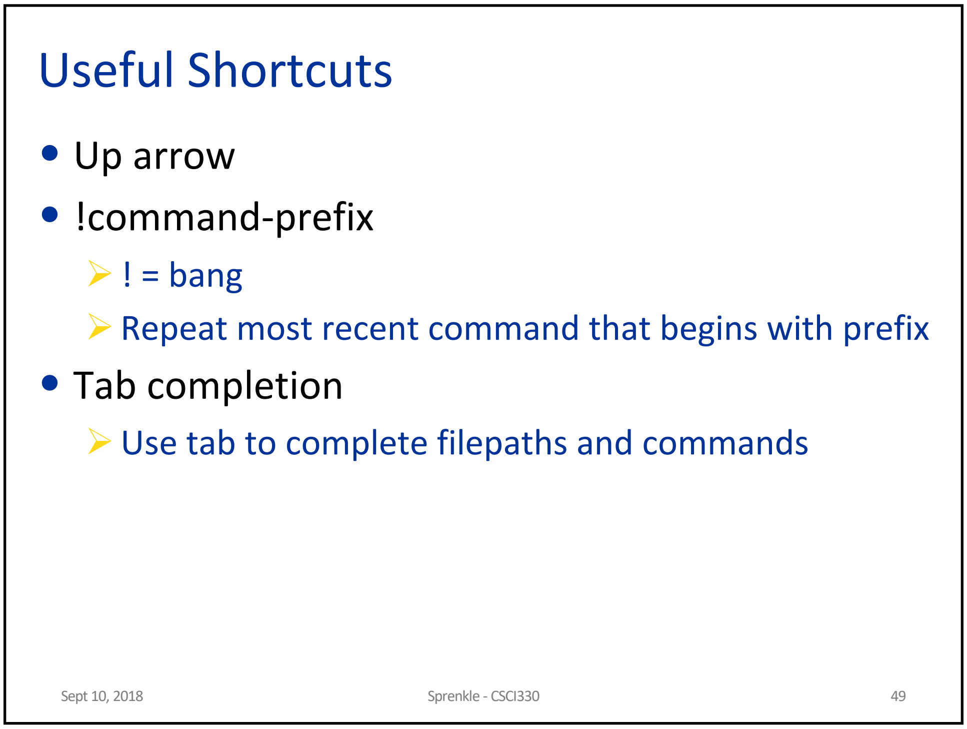
Handout Discussion

- What additional Unix commands did you find?
- What are the tradeoffs to the Unix command design (many small, simple programs; can be combined)?

Unix Design

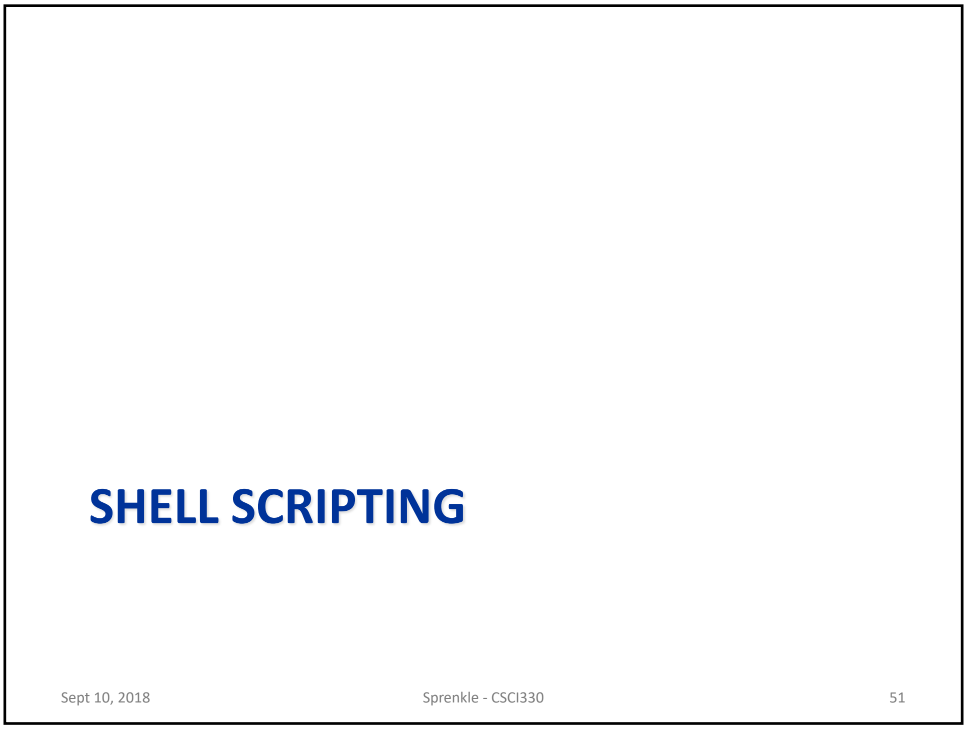
- Small, simple programs
  - Easier to maintain
  - Single-responsibility principle
- Combine (a few or lots) with pipes
  - Easy to combine with a simple interface |
- Not-so-user-friendly to get started

USEFUL SHORTCUTS



Useful Shortcuts

- Up arrow
- !command-prefix
  - ! = bang
  - Repeat most recent command that begins with prefix
- Tab completion
  - Use tab to complete filepaths and commands



SHELL SCRIPTING

Review: What is a Shell?

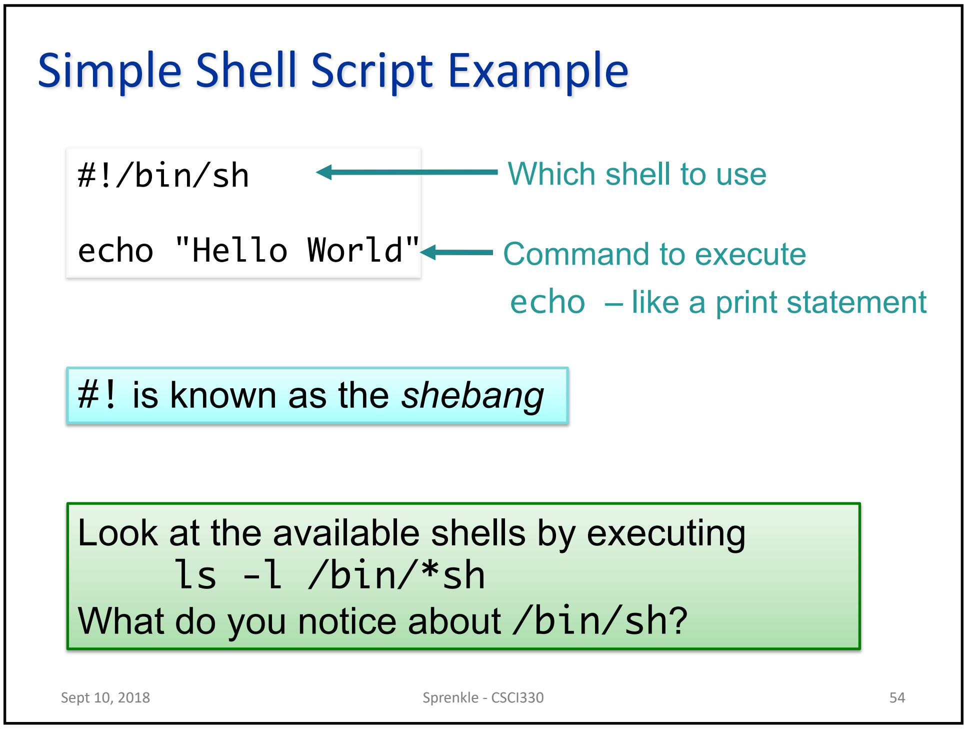
- User interface to the operating system
- Command-line interpreter
- Functionality:
  - Execute other programs
  - Manage files
  - Manage processes
- A program like any other
- Basic form of shell:

```
while <read command>:
    parse command
    execute command
```

hides details of underlying operating system

What is a shell script?

- A *shell script* is a list of commands to be run by a shell
  - basically a program
  - uses shell commands instead of C or Java statements
- Why?
  - automate repetitious tasks
    - Ex: executing a program on a large set of test inputs
  - package up commonly executed command sequences
  - create our own commands



Simple Shell Script Example

```
#!/bin/sh

echo "Hello World"
```

Which shell to use

Command to execute

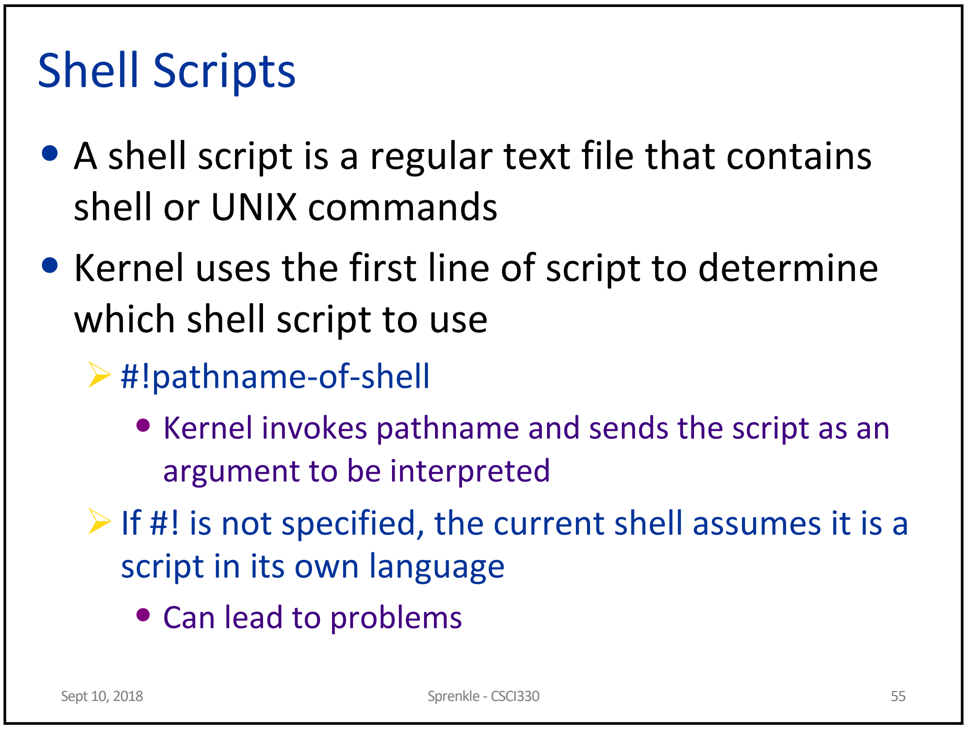
`echo` – like a print statement

`#!` is known as the *shebang*

Look at the available shells by executing

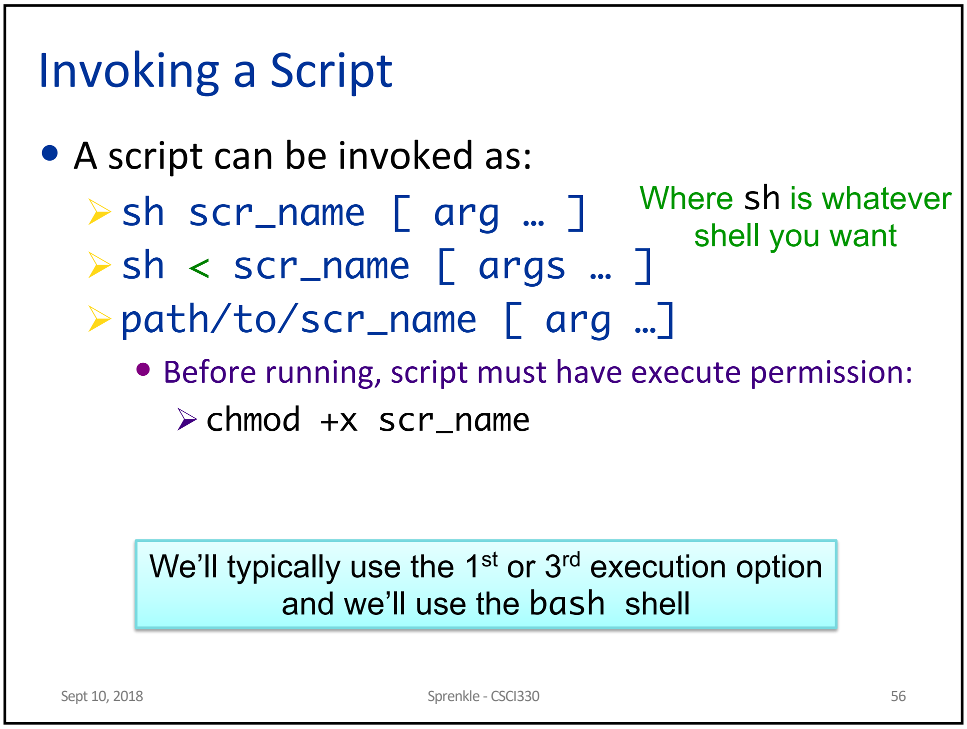
```
    ls -l /bin/*sh
```

What do you notice about `/bin/sh`?



Shell Scripts

- A shell script is a regular text file that contains shell or UNIX commands
- Kernel uses the first line of script to determine which shell script to use
  - #!pathname-of-shell
    - Kernel invokes pathname and sends the script as an argument to be interpreted
  - If #! is not specified, the current shell assumes it is a script in its own language
    - Can lead to problems

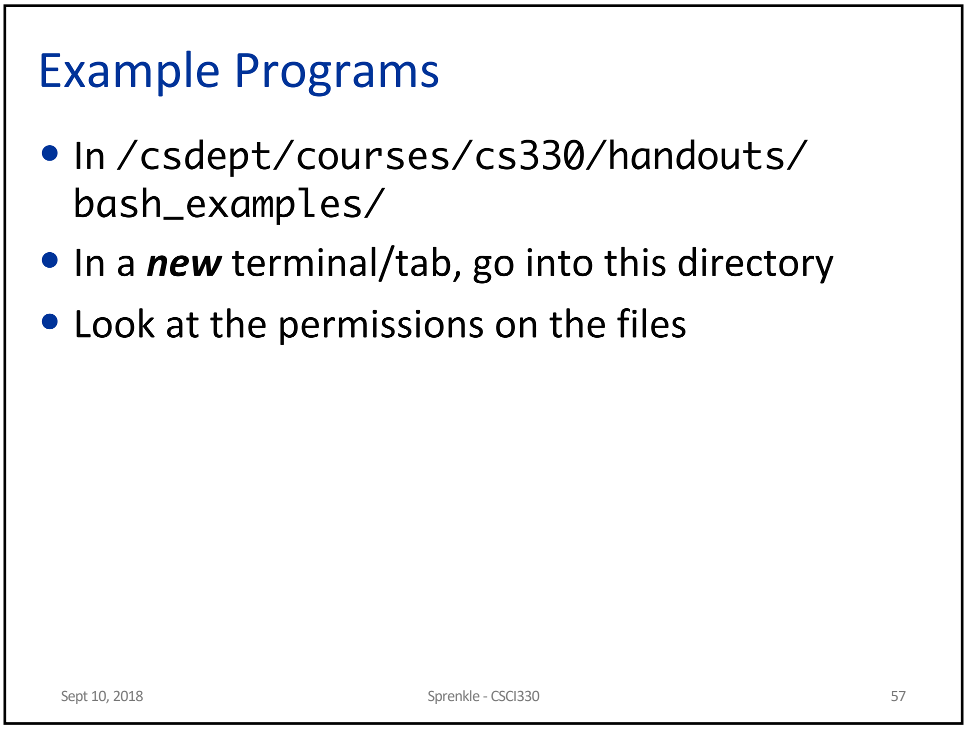


Invoking a Script

- A script can be invoked as:
  - ➢ `sh scr_name [ arg … ]`
  - ➢ `sh < scr_name [ args … ]`
  - ➢ `path/to/scr_name [ arg …]`
    - Before running, script must have execute permission:
      - ➢ `chmod +x scr_name`

Where `sh` is whatever shell you want

We'll typically use the 1st or 3rd execution option and we'll use the `bash` shell



Example Programs

- In `/csdept/courses/cs330/handouts/bash_examples/`
- In a ***new*** terminal/tab, go into this directory
- Look at the permissions on the files

Writing Your First Bash Script

- **Bash**: **B**ourne-**a**gain **sh**ell
  - Unix shell and command language
- Open your favorite text editor
- Write a simple bash script:
  - Type in the shebang `#!/bin/sh`
  - And the command:

    ```
    echo "Hello World"
    ```
  - and save as `hello.sh`
- Type `bash hello.sh` to run

Comments

- Comments begin with a #
- Comments end at the end of the line
- Comments can begin whenever a token begins
- Many text editors will help you with syntax highlighting
- Examples:

```
# This is a comment
# and so is this
grep foo bar # this is a comment
grep foo bar# this is not a comment
```

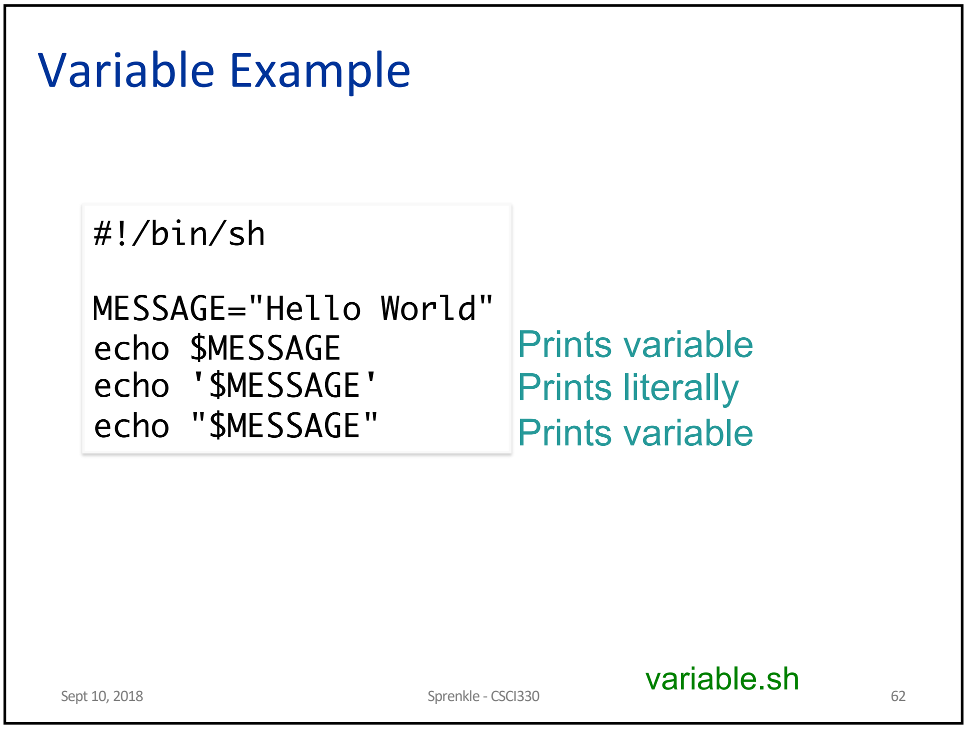
Style requirement:
A comment on 2nd line in your script that lists you as author

Your Second Script

- Write a script that
  - Displays the files in the current directory
  - Lists all logged-in users
- Your script should contain authorship info near the top
- Build in pieces (Yes, even this short script)
- Execute and test your script
  - Verify the output

Variables

- Don't have to be declared in advance
- Untyped: the same variable can hold an integer value or a string
- Syntax for using variables (bash):
  - Defining the value of a variable name:
    - `name=value` ← Notice no spaces around =
  - Using the variable name:
    - `$name` or `${name}`
- Variables can be local or environment
  - Environment variables are part of UNIX and can be accessed by child processes

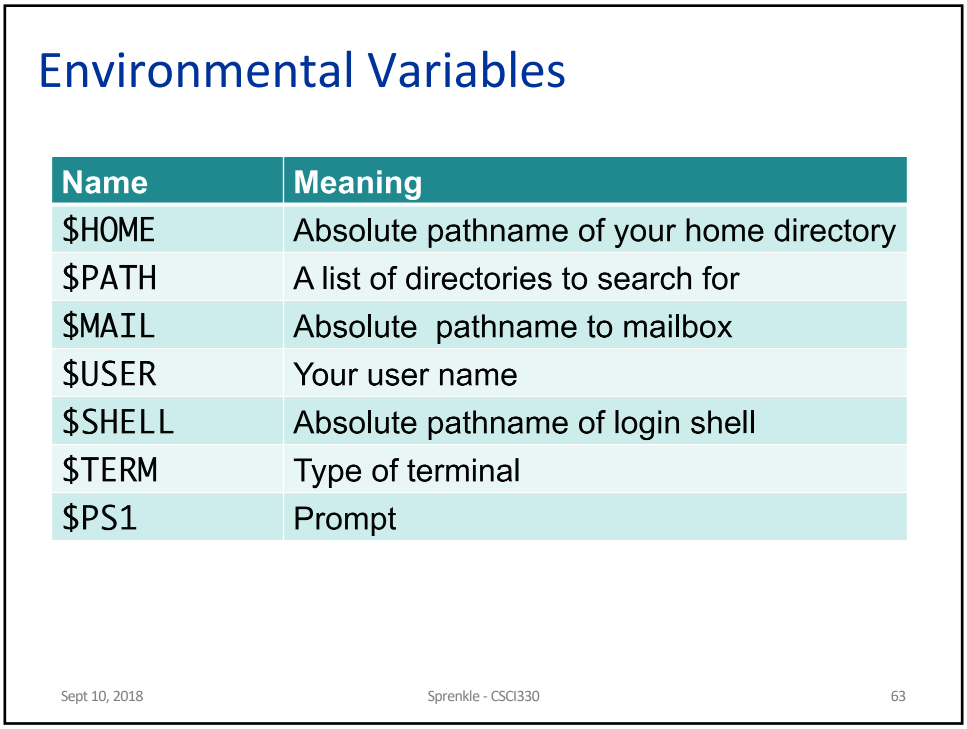


Variable Example

```
#!/bin/sh

MESSAGE="Hello World"
echo $MESSAGE
echo '$MESSAGE'
echo "$MESSAGE"
```

Prints variable
Prints literally
Prints variable

variable.sh



Environmental Variables

Name	Meaning
$HOME	Absolute pathname of your home directory
$PATH	A list of directories to search for
$MAIL	Absolute pathname to mailbox
$USER	Your user name
$SHELL	Absolute pathname of login shell
$TERM	Type of terminal
$PS1	Prompt

Using Environment Variables

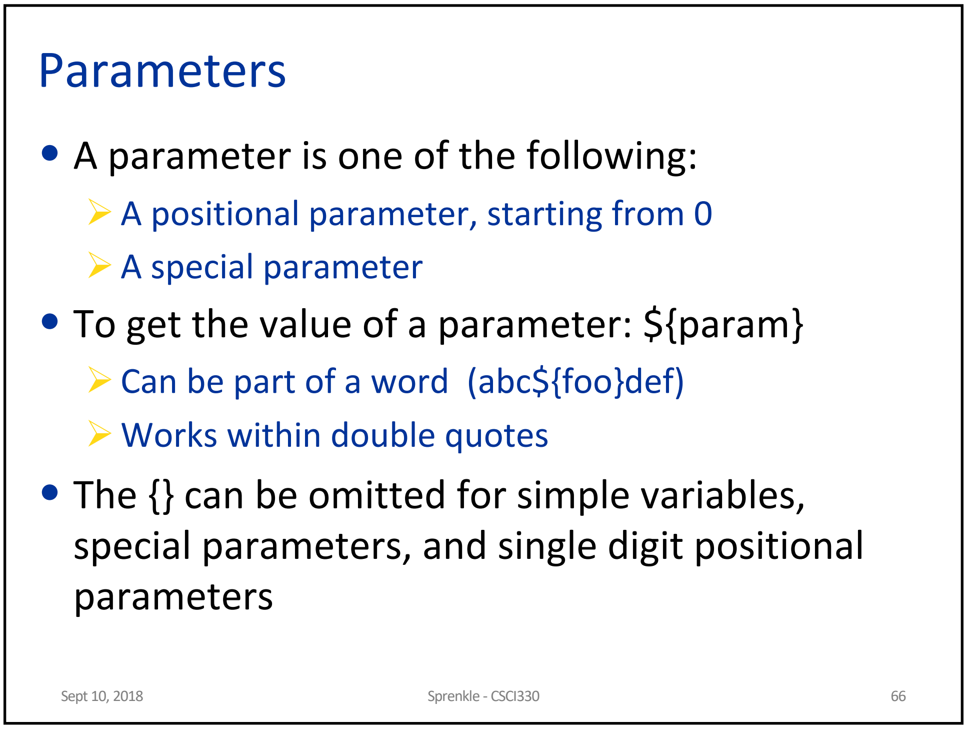
```
#!/bin/bash

echo I am $USER
echo "I live at $HOME"
```

Both echo statements work with or without quotes

env_var.sh

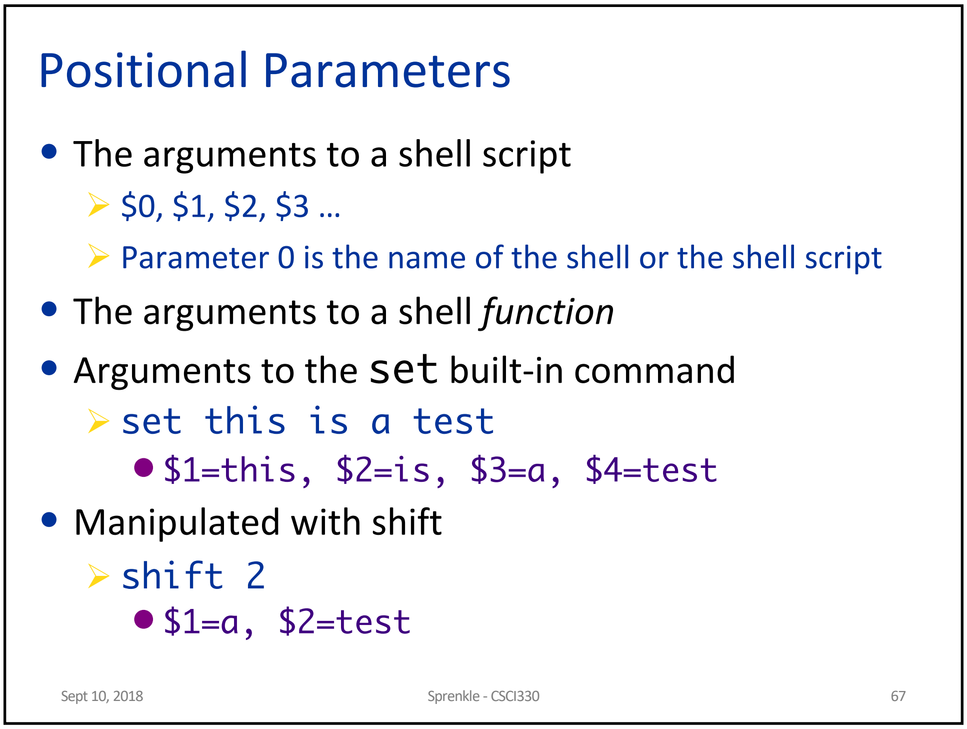
Modify your second script

- Write a script that
  - Displays the files in YOUR HOME directory
  - Lists all logged-in users
- Your script should contain authorship info near the top
- Build in pieces (Yes, even this short script)
- Execute and test your script
  - Verify the output



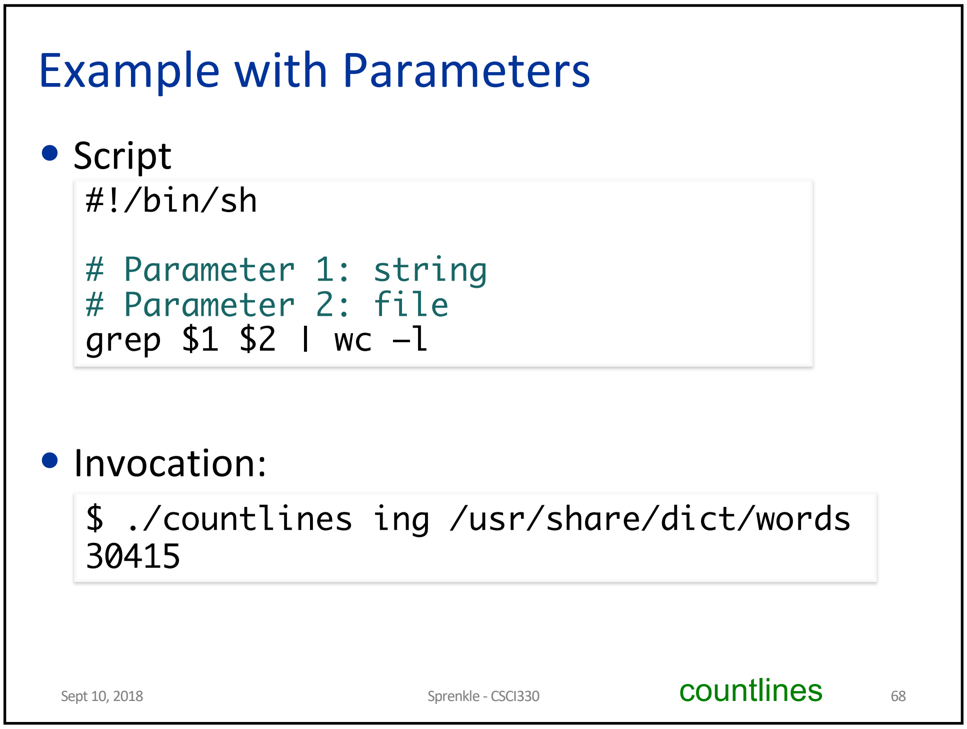
Parameters

- A parameter is one of the following:
  - A positional parameter, starting from 0
  - A special parameter
- To get the value of a parameter: ${param}
  - Can be part of a word (abc${foo}def)
  - Works within double quotes
- The {} can be omitted for simple variables, special parameters, and single digit positional parameters



Positional Parameters

- The arguments to a shell script
  - $0, $1, $2, $3 …
  - Parameter 0 is the name of the shell or the shell script
- The arguments to a shell *function*
- Arguments to the `set` built-in command
  - `set this is a test`
    - `$1=this, $2=is, $3=a, $4=test`
- Manipulated with shift
  - `shift 2`
    - `$1=a, $2=test`



Example with Parameters

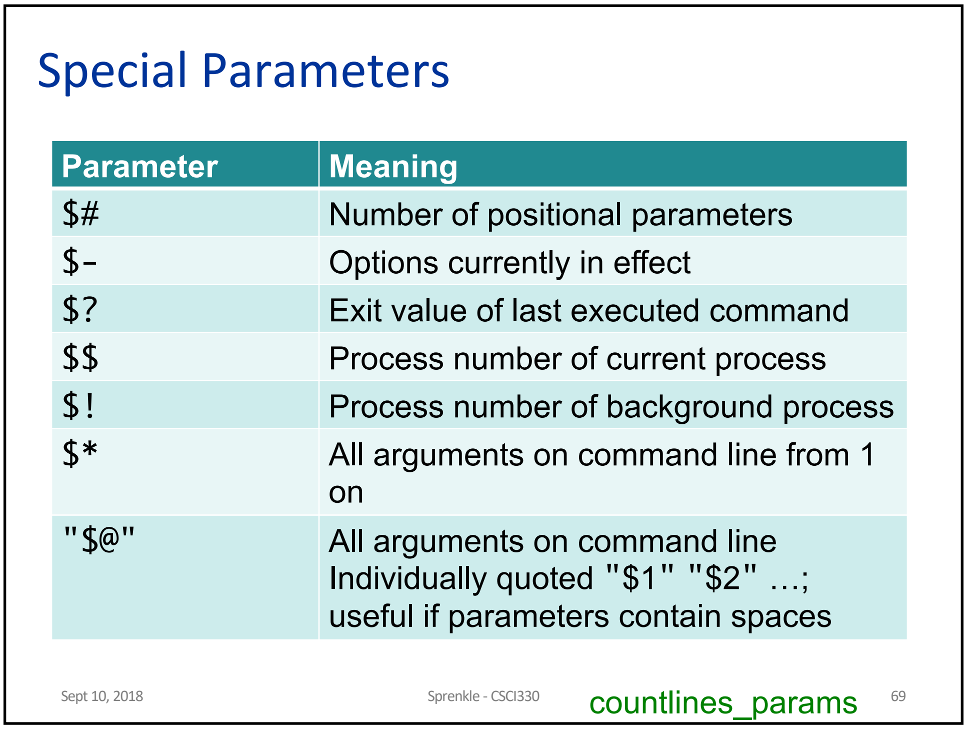
- Script

```
#!/bin/sh

# Parameter 1: string
# Parameter 2: file
grep $1 $2 | wc –l
```

- Invocation:

```
$ ./countlines ing /usr/share/dict/words
30415
```

countlines



Special Parameters

Parameter	Meaning
$#	Number of positional parameters
$-	Options currently in effect
$?	Exit value of last executed command
$$	Process number of current process
$!	Process number of background process
$*	All arguments on command line from 1 on
"$@"	All arguments on command line Individually quoted "$1" "$2" …; useful if parameters contain spaces

countlines_params

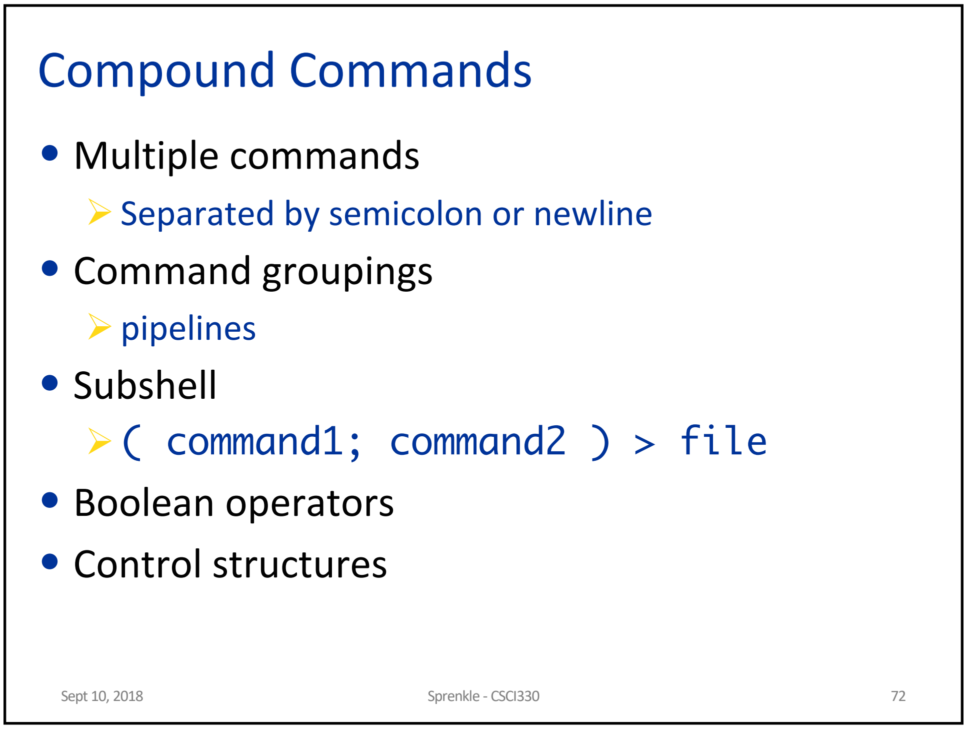
Special Characters

- The shell processes the following characters specially unless quoted:
  - | & ( ) < > ; " ' $ ` space tab newline
- The following are special whenever patterns are processed:
  - * ? [ ]
- The following are special at the beginning of a word:
  - # ~
- The following is special when processing assignments:
  - =

Command Substitution: ``

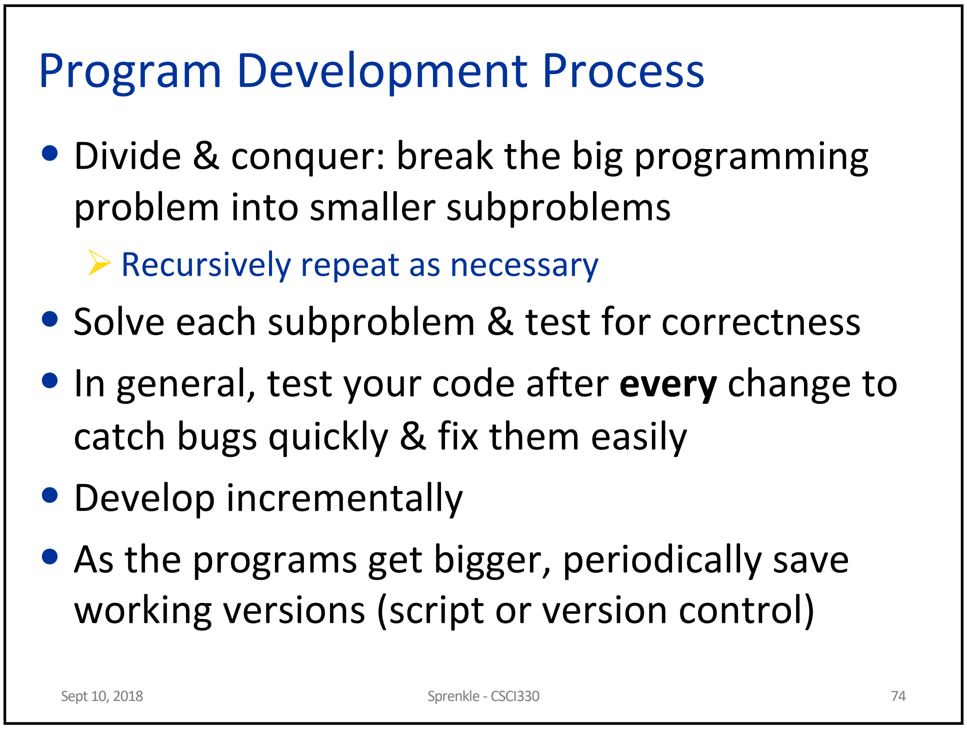
- Used to turn the output of a command into a string
- Used to create arguments or variables

```
$ date
Mon Sep 10 11:46:37 EDT 2018
$ NOW=`date`
$ echo $NOW
Mon Sep 10 11:46:37 EDT 2018
$ PATH=`myscript`:$PATH
```



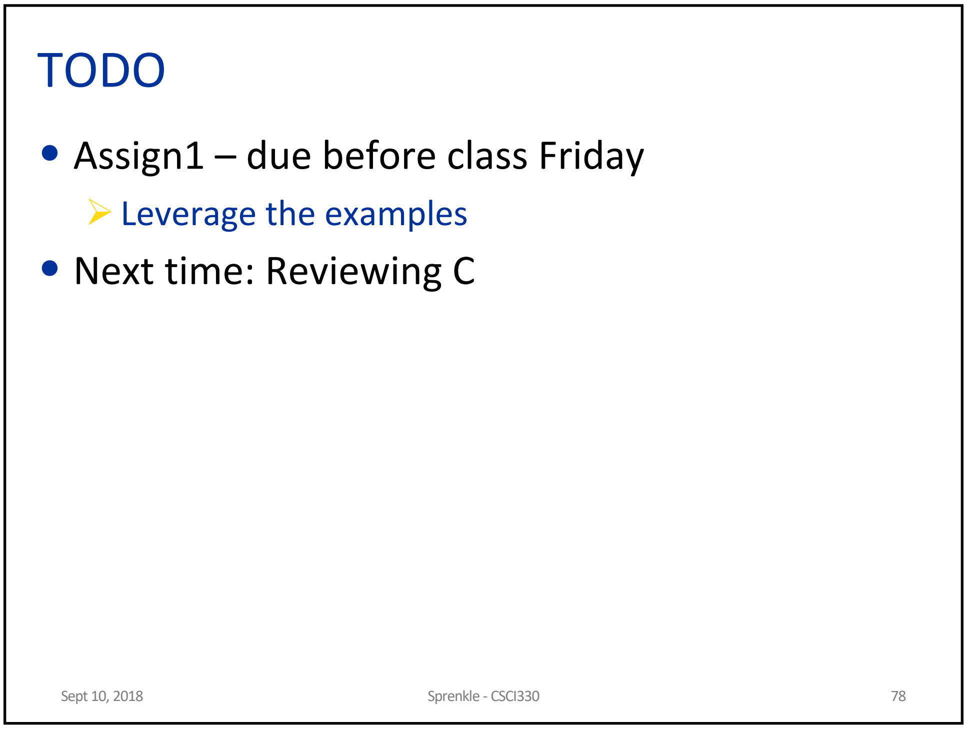
Compound Commands

- Multiple commands
  - Separated by semicolon or newline
- Command groupings
  - pipelines
- Subshell
  - `( command1; command2 ) > file`
- Boolean operators
- Control structures



Program Development Process

- Divide & conquer: break the big programming problem into smaller subproblems
  - Recursively repeat as necessary
- Solve each subproblem & test for correctness
- In general, test your code after **every** change to catch bugs quickly & fix them easily
- Develop incrementally
- As the programs get bigger, periodically save working versions (script or version control)



TODO

- Assign1 – due before class Friday
  - Leverage the examples
- Next time: Reviewing C