The Missing Semester:
Linux Distros, Shell Tools and Scripting
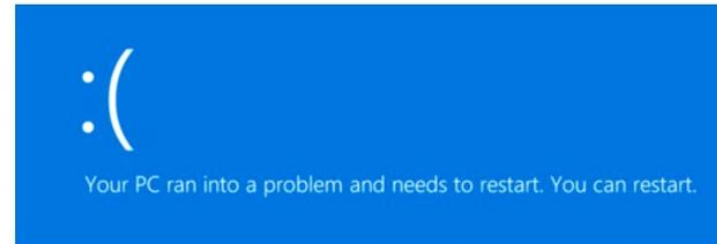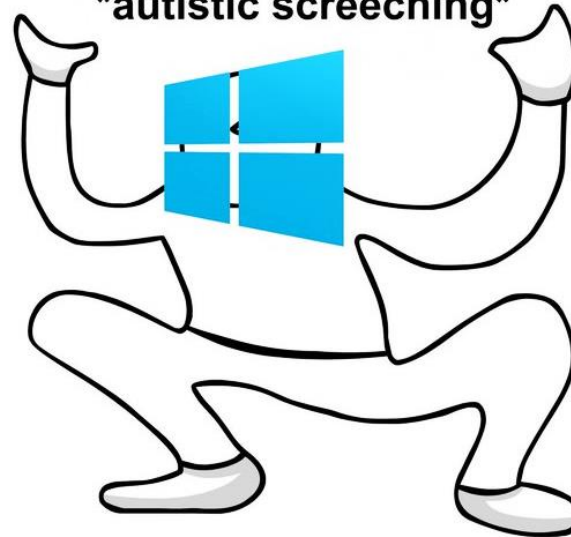
Why Linux when I have Windows?

1. Linux is free and that's a great price!
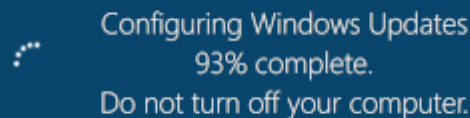2. You're Free To Tinker
3. There's Support For Most Programming Languages
4. Feel The Power Of Bash Scripting
5. Linux Expertise Is A Desirable Skill
6. Privacy, Control and Support

Configuring Windows Updates
93% complete.
Do not turn off your computer.

Seriously, Why Would You Use Anything Else?

acm Sri Venkateswara College of Engineering

The Missing Semester: Choosing your Linux Distro

How are distros different from each other?

Although all Linux distributions are based on the Linux kernel, each offers a set of major differentiating criteria that specialize them:

•**Architecture**:
The type of chips that the distribution supports, through the included kernel.

•**Initial Software**:
The initial set of softwares it's shipped with.

How are distros different from each other?

- **Package Manager**: The default package-management tool for the distribution.

- **Desktop Manager**: The graphical user interface for the distribution.

Package Managers

- **dpkg**: Manages Debian-specific (.DEB) packages in Debian-based distribuition including Ubuntu and Linux Mint—through tools like APT.

- **RPM Package Manager**: Installs/manages Redhat Package Manager (.RPM) packages. Uses tools like DNF, yum and zypper.

Package Managers

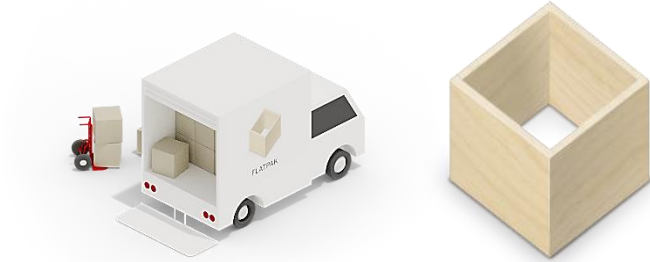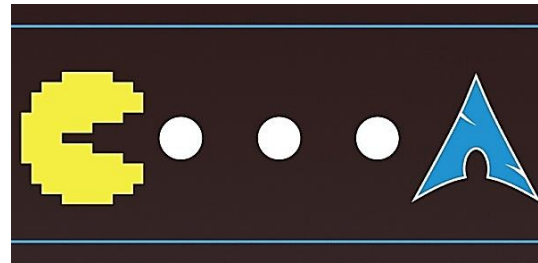- **flatpak**: A sandboxed/containered format that's cross-platform.

- **pacman**: Common in Arch Linux and its derivatives.

- **snap**: A Ubuntu-specific form of containerized application deployment.

The Missing Semester: Shell and Scripting

Shell

- Computers these days have a variety of interfaces for giving them commands; **fancy graphical user interfaces**, voice interfaces, and even AR/VR are everywhere.

- These are great for 80% of use-cases, but they are often **fundamentally restricted** in what they allow you to do — you cannot press a button that isn't there or give a voice command that hasn't been programmed.

- To take **full advantage of the tools your computer provides**, we have to go old-school and drop down to a textual interface: The Shell.

Shell – Sample Commands

Date program: prints the current date and time.

```
student@svceacm-playground:~:~$ date
Fri 10 Jan 2020 11:49:31 AM EST
```

The echo program simply prints out its arguments. The shell parses the command by splitting it by whitespace, and then runs the program indicated by the first word, supplying each subsequent word as an argument that the program can access.

```
student@svceacm-playground:~$ echo hello
hello
```

BASH
THE BOURNE-AGAIN SHELL

How shell works?

If you ever want more information about a program's arguments, inputs, outputs, or how it works in general, give the man program a try. It takes as an argument the name of a program, and shows you its manual page. Press q to exit.

```
student@svceacm-playground:~$ man ls
```

How shell works?

But how does the shell know how to find the date or echo programs? Well, the shell is a programming environment, just like Python or Ruby, and so it has variables, conditionals, loops, and functions. When you run commands in your shell, you are really writing a small bit of code that your shell interprets.

If the shell is asked to execute a command that doesn't match one of its programming keywords, it consults an environment variable called $PATH that lists which directories the shell should search for programs when it is given a command:

```
student@svceacm-playground:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

student@svceacm-playground:~$ which echo
/bin/echo

student@svceacm-playground:~$ /bin/echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Sri Venkateswara
College of Engineering

Navigating in the shell

```
student@svceacm-playground:~$ pwd
/home/missing

student@svceacm-playground:~$ cd /home
student@svceacm-playground:/home$ pwd
/home

student@svceacm-playground:/home$ cd ..
student@svceacm-playground:/$ pwd
/

student@svceacm-playground:/$ cd ./home
student@svceacm-playground:/home$ pwd
/home

student@svceacm-playground :/home$ cd student
student@svceacm-playground:~$ pwd
/home/student
```

```
student@svceacm-playground:~$ ls
student@svceacm-playground:~$ cd ..
student@svceacm-playground:/home$ ls
Student

student@svceacm-playground:/home$ cd ..

student@svceacm-playground :/$ ls
bin
boot
dev
etc
home
...
```

Sri Venkateswara
College of Engineering

Connecting Programs

- In the shell, programs have two primary "streams" associated with them: **their input stream and their output stream**.

- When the program tries to read input, it **reads from the input stream**, and when it prints something, it **prints to its output stream**.

- Normally, a program's input and output are both your terminal. That is, your keyboard as input and your screen as output.

- The simplest form of redirection is < file and > file.

```
student@svceacm-playground:~$ echo hello > hello.txt
student@svceacm-playground:~$ cat hello.txt
hello
student@svceacm-playground:~$ cat < hello.txt
hello
student@svceacm-playground:~$ cat < hello.txt > hello2.txt
student@svceacm-playground:~$ cat hello2.txt
hello
```

Sri Venkateswara
College of Engineering

Pipelining commands

The | operator lets you "chain" programs such that the output of one is the input of another:

```
student@svceacm-playground:~$ curl --head --silent google.com | grep --ignore-
case content-length | cut --delimiter=' ' -f2
219
```

Sri Venkateswara
College of Engineering