

# Recitation 1

Computer Architecture (section 1)

# About Me

- Name: Alborz Jelvani
- Email: [alborz.jelvani@rutgers.edu](mailto:alborz.jelvani@rutgers.edu)
- Office Hours: Fridays 3:30pm - 4:30pm, CORE 333 and Zoom
- Recitations: Friday 2:15pm-3:10pm, BE-250
  - Attendance taken via Canvas quiz

# How to make a Rutgers CS account

- <https://services.cs.rutgers.edu/accounts/>



## Account Management Tools for Computer Science Systems

Click this

**Note:** Your computer science user ID is your **University NetID**. You must have a University NetID before you can activate a computer science account. Here's the University's tool for activating your NetID: <https://netid.rutgers.edu>

Computer science systems use passwords that are separate from your University password. You can make them the same, but it is somewhat better from a security point of view if you use different passwords.

- **Lifetime of accounts and files**

When users leave the University (or computer science), their accounts are closed. Files are archived. They will be deleted after a year, except for faculty files. Shared directories will be deleted or archived based on the user that owns them.

Sometimes users will have a continuing association with the department even after leaving. Accounts may be continued as guest or retiree accounts. Any faculty member can sponsor a guest. The Computer Science Department handles retirees.

- [Activate an account on a Computer Science system](#)

This will let you create an account for Computer Science Department systems. Computer science faculty, majors, grad students, and students enrolled in computer science courses other than 110, 170 and 494 are eligible for accounts.

- [Set or reset your Computer Science password.](#)

This will show you a University login screen. So as long as you remember your University password, you set or reset your CS password. If you need more security than a simple password can afford, please see [two factor authentication](#).

- [Group and Guest management.](#) This will you create and manage groups. This is useful for three purposes:

- If you want to share files with a group of people, you can create a group that lists the people and then change the permissions of the file so they can access it. For more help with this, see [Sharing files](#).
- If you are authorized (normally faculty) and want to allow grad students or collaborators to access research or instructional systems, you can create a guest group. For help with this, see [Managing your guest users](#).
- If you are running your own computer cluster, you can use groups to control what users are allowed to login.

- Consider giving us contact information, in case we need to reach you for support reasons.

You can set additional information using "ipa user-mod" on any of our systems, e.g. "ipa user-mod YOURNETID --phone=8484452001". "ipa user-mod --help" will give a list of what can be set. "ipa user-show YOURNETID" will show your current information. You only need to do this once. The same information applies to all of our systems. Note that changing first name, lastname, and GECOS won't do anything useful, since we reset them to the official University names nightly.

# Weblogin for Rutgers CS

- <https://weblogin.cs.rutgers.edu/>
  - Login to Rutgers CS account
  - Pick a machine
  
- <https://report.cs.rutgers.edu/nagiosnotes/iLab-machines.html>
  - View machine status

## RECENT CONNECTIONS



kill.cs.rutgers.edu



prolog.cs.rutgers.edu



i-lab1.cs.rutgers.edu

## ALL CONNECTIONS

Choose a host to connect to from the menu, or type a hostname in this box:

Enter Connection URI:

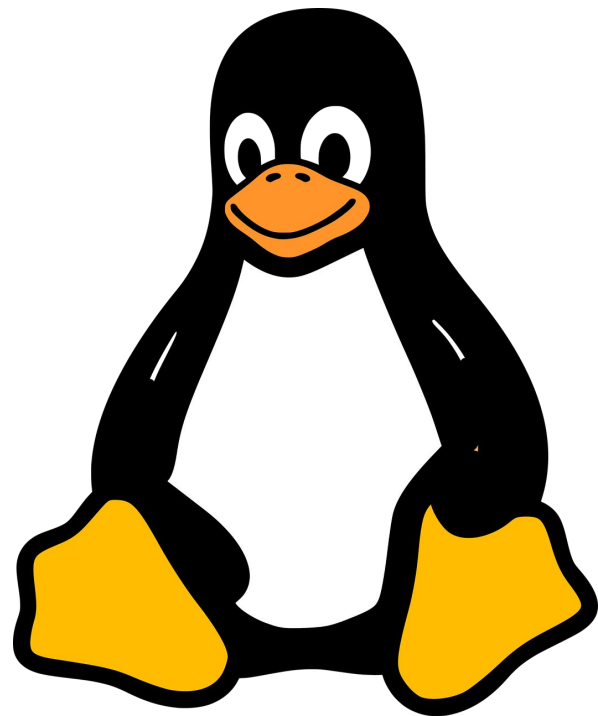
- geneva
- i-lab1.cs.rutgers.edu
- i-lab2.cs.rutgers.edu
- i-lab3.cs.rutgers.edu
- i-lab4.cs.rutgers.edu
- kill-all-my-sessions
- assembly.cs.rutgers.edu
- basic.cs.rutgers.edu
- batch.cs.rutgers.edu
- boole.cs.rutgers.edu
- butter.cs.rutgers.edu
- c211-1.cs.rutgers.edu
- c211-2.cs.rutgers.edu
- c211-3.cs.rutgers.edu
- c211-11.cs.rutgers.edu
- c211-12.cs.rutgers.edu
- c211-13.cs.rutgers.edu
- c246-1.cs.rutgers.edu
- c246-2.cs.rutgers.edu
- c329-1.cs.rutgers.edu
- c331-1.cs.rutgers.edu

- cheese.cs.rutgers.edu
- cp.cs.rutgers.edu
- cpp.cs.rutgers.edu
- crayon.cs.rutgers.edu
- data7.cs.rutgers.edu
- dogmatic.rutgers.edu
- frost.cs.rutgers.edu
- grep.cs.rutgers.edu
- h206-2.cs.rutgers.edu
- h257-1.cs.rutgers.edu
- h266-1.cs.rutgers.edu
- h273-1.cs.rutgers.edu
- h275-1.cs.rutgers.edu
- h275-2.cs.rutgers.edu
- h275-g4.cs.rutgers.edu
- h405-2.cs.rutgers.edu
- h410-1.cs.rutgers.edu
- h410-2.cs.rutgers.edu
- h412-2.cs.rutgers.edu
- h414-1.cs.rutgers.edu
- h414-2.cs.rutgers.edu

- java.cs.rutgers.edu
- kill.cs.rutgers.edu
- kilaatu.rutgers.edu
- kleene.cs.rutgers.edu
- kolmogorov.cs.rutgers.edu
- less.cs.rutgers.edu
- lisp.cs.rutgers.edu
- ls.cs.rutgers.edu
- man.cs.rutgers.edu
- mv.cs.rutgers.edu
- pascal.cs.rutgers.edu
- perl.cs.rutgers.edu
- plastic.cs.rutgers.edu
- popsicle.cs.rutgers.edu
- post.cs.rutgers.edu
- prolog.cs.rutgers.edu
- pwd.cs.rutgers.edu
- python.cs.rutgers.edu
- rlab1.cs.rutgers.edu
- rlab2.cs.rutgers.edu
- rlab3.cs.rutgers.edu

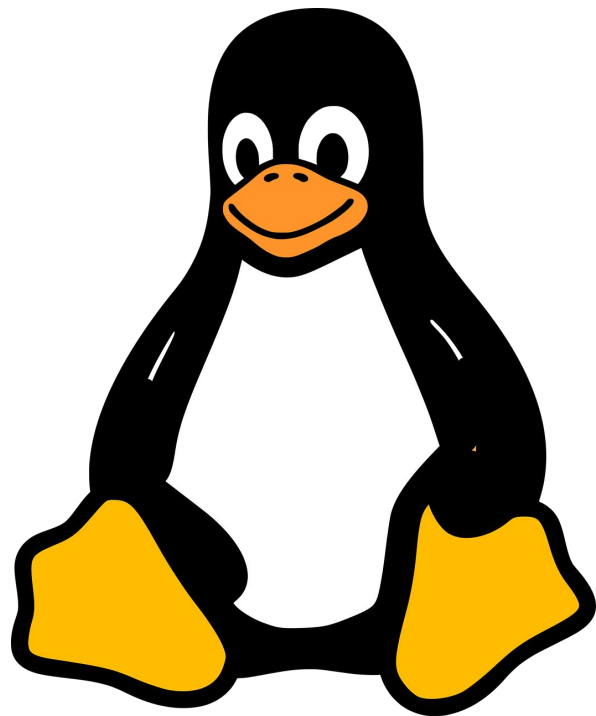
# Linux Shell Intro

- Interaction with the OS occurs through the *shell*.



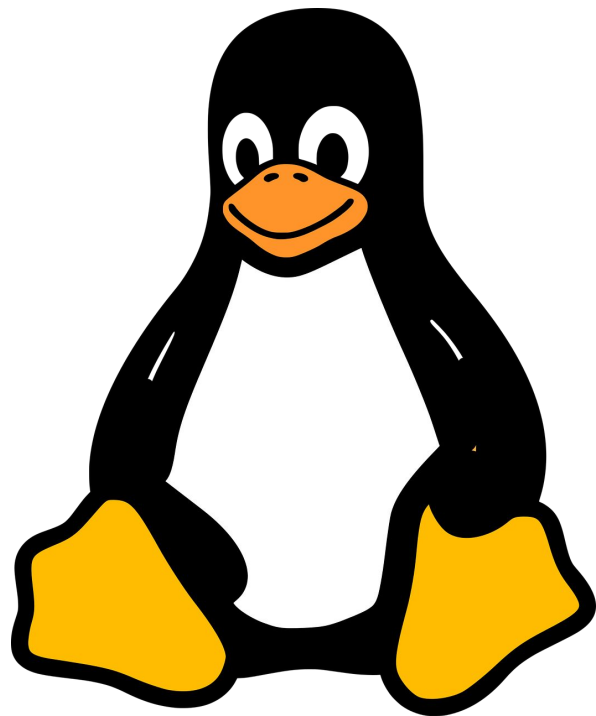
# Linux Shell Intro

- Interaction with the OS occurs through the *shell*.
- The *shell* interprets commands and prints responses.
  - The default shell in Linux is a program called **bash**.

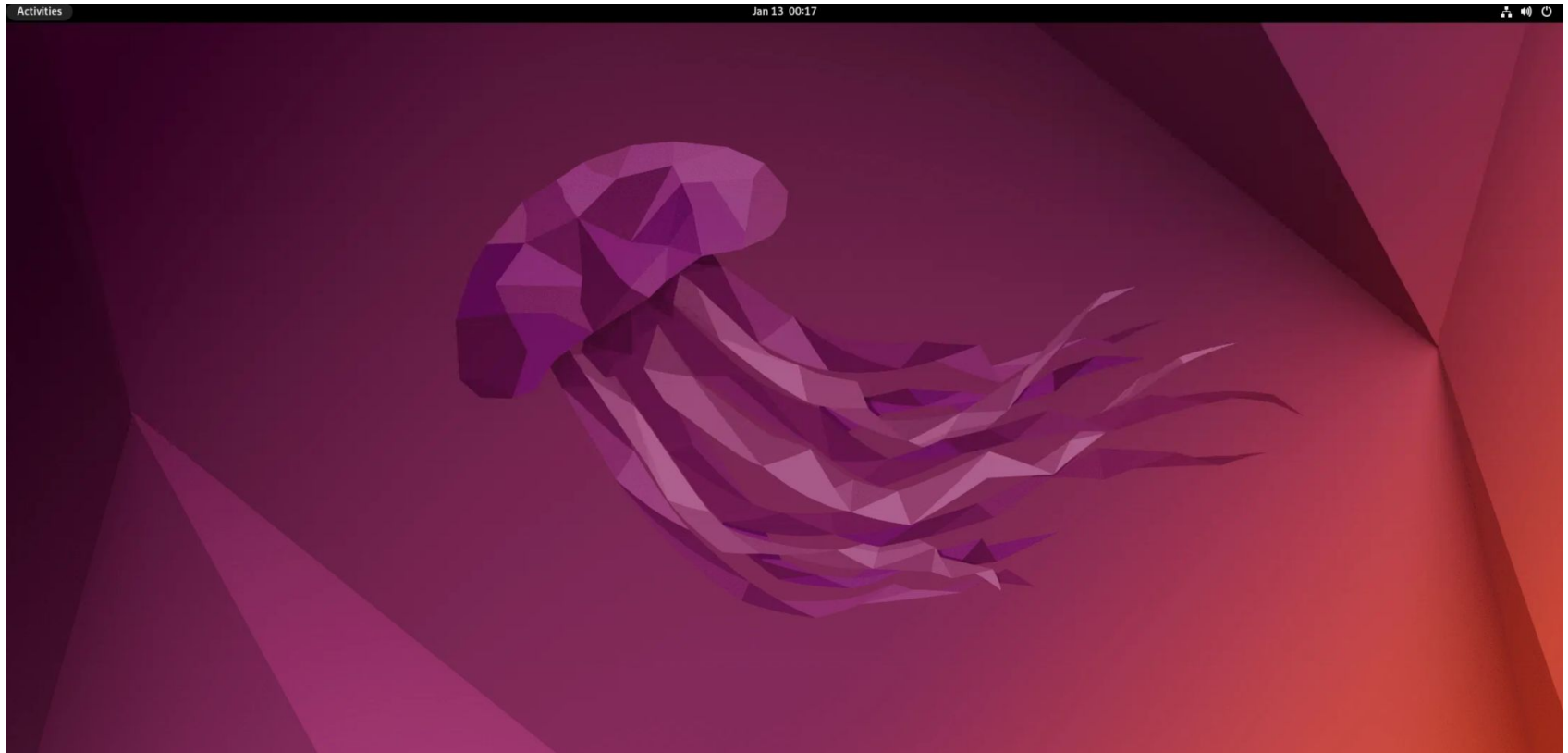


# Linux Shell Intro

- Interaction with the OS occurs through the *shell*.
- The *shell* interprets commands and prints responses.
  - The default shell in Linux is a program called **bash**.
- A *terminal emulator* (AKA terminal) is a program that provides an interface between the user and a shell.

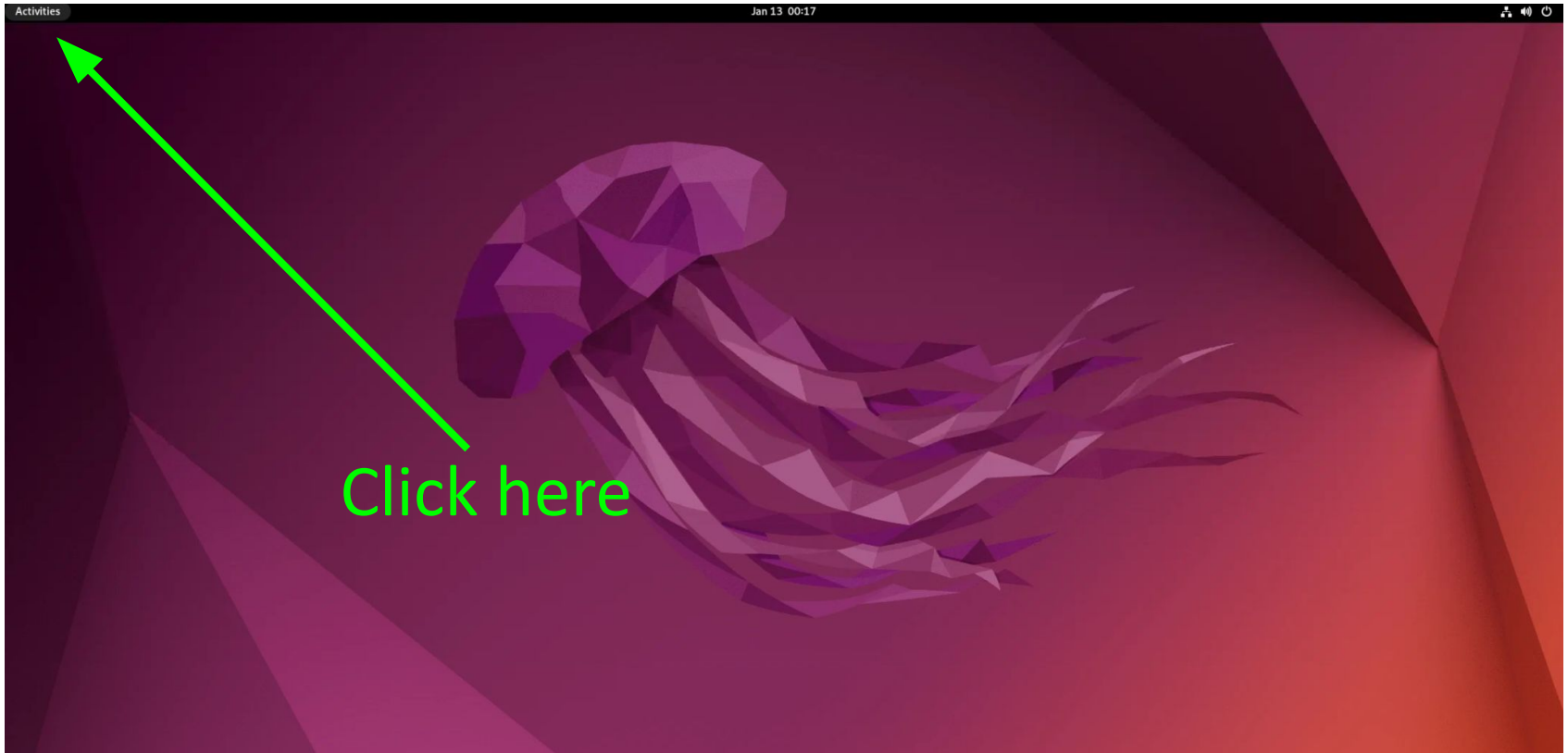


# Opening a Terminal on iLab

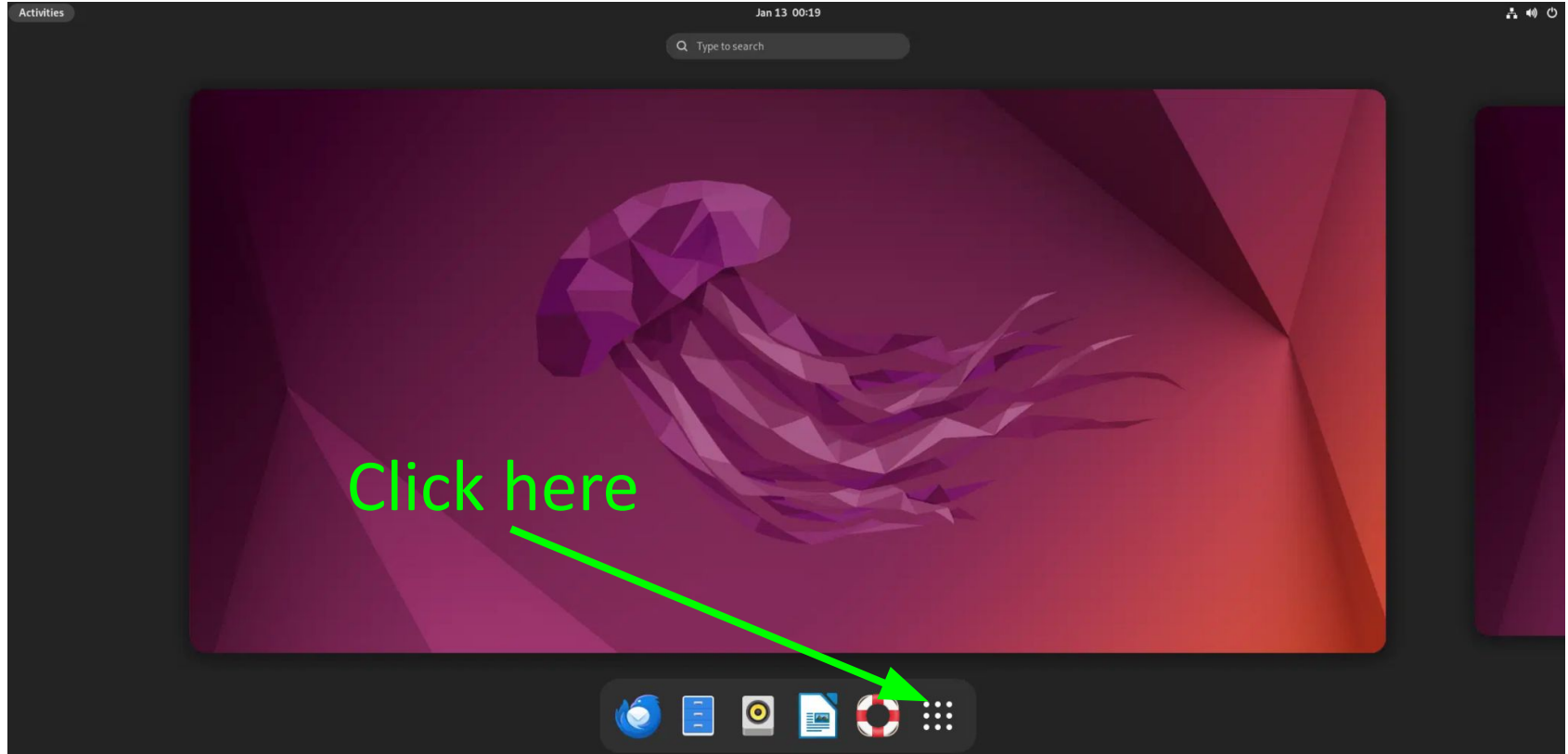




# Opening a Terminal on iLab

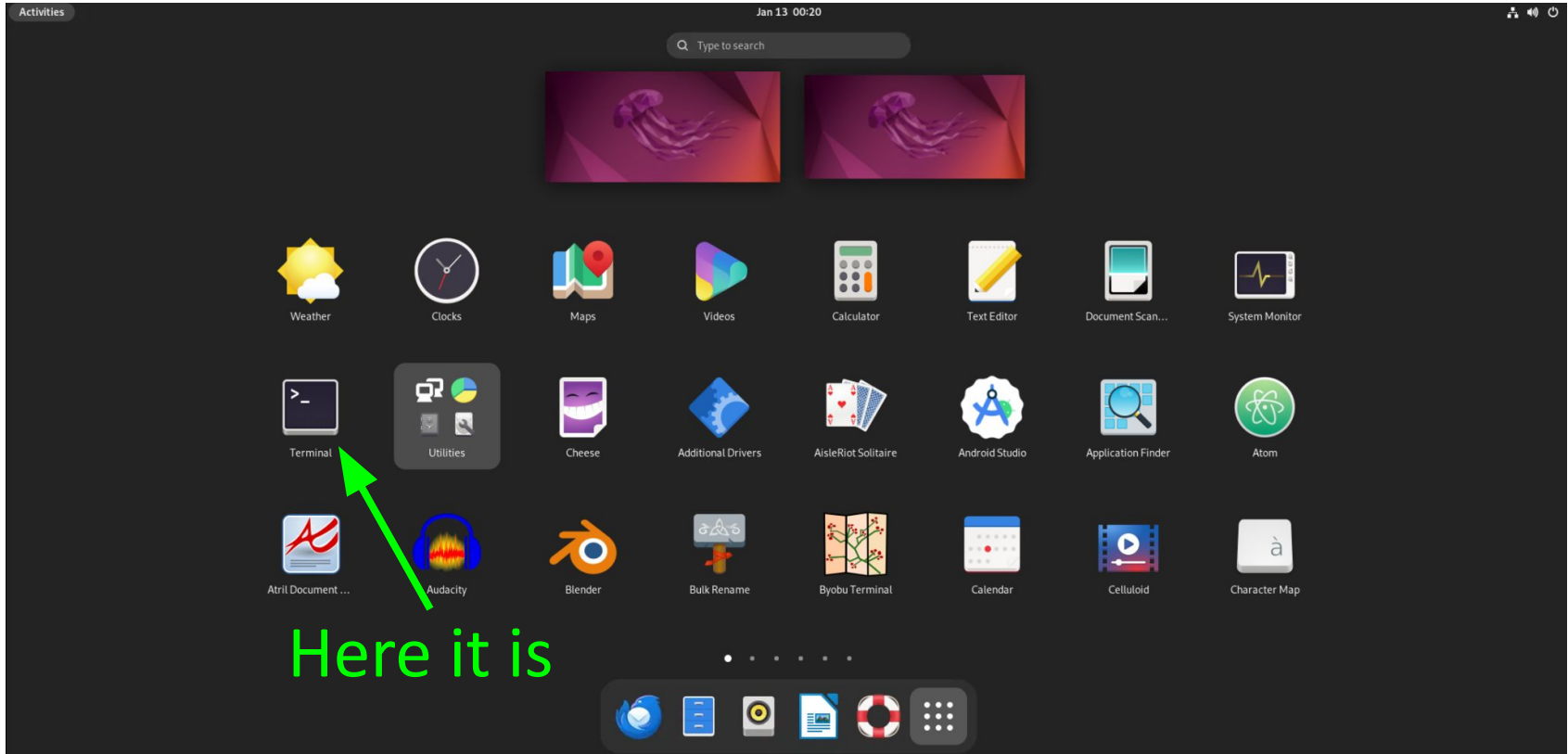


# Opening a Terminal in Ubuntu



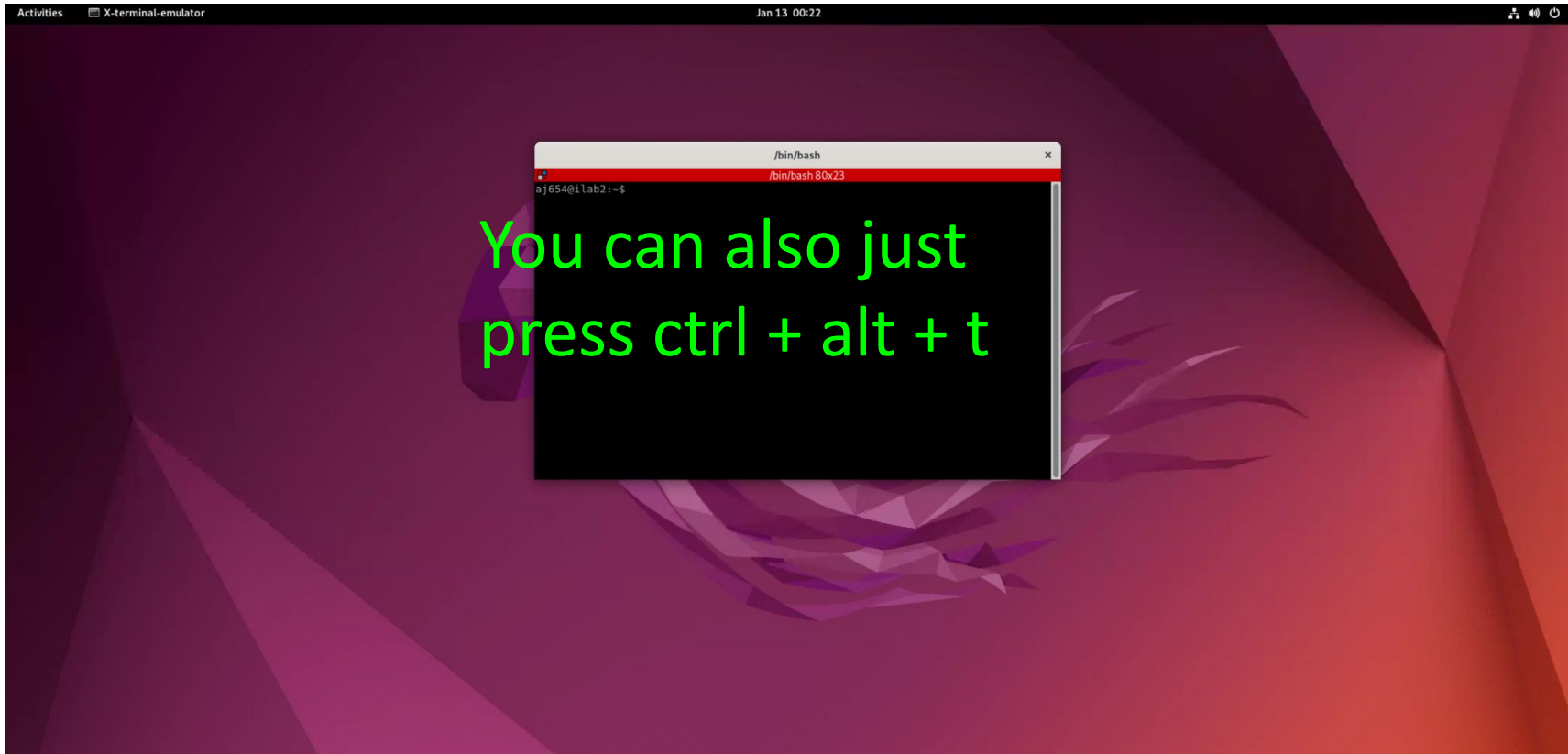
Click here

# Opening a Terminal in Ubuntu



Here it is

# Opening a Terminal on iLab



# Linux Shell Anatomy

```
aj654@ilab1:~/someDirectory$ <CMD HERE>
```

# Linux Shell Anatomy

```
aj654@ilab1:~/someDirectory$ <CMD HERE>
```

*Current user*



# Linux Shell Anatomy

```
aj654@ilab1:~/someDirectory$ <CMD HERE>
```

*Current user*

*System name*

# Linux Shell Anatomy

```
aj654@milab1:~/someDirectory$ <CMD HERE>
```

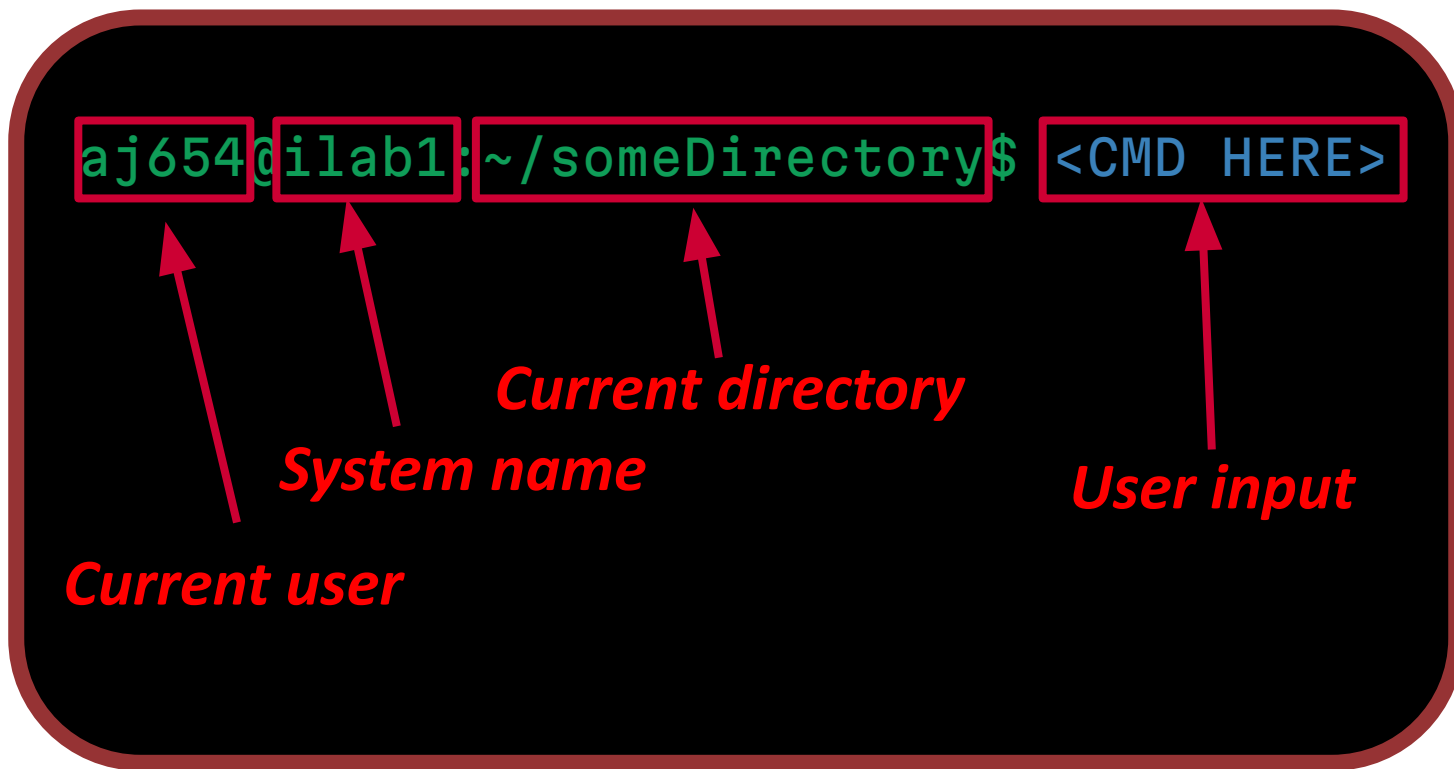
*Current user*

*System name*

*Current directory*



# Linux Shell Anatomy



# Some Linux Commands

**cd** - change directory

**ls** - list files in directory

**mkdir** - create directory

**touch** - used to create files

**rm** - removes a file or directory

**sudo** - superuser do, do not use this on ilabs

**pwd** - print current working directory

**whoami** - view current user

**echo** - print inputted text to screen

**man** - view manual page for a command

**cat** - print contents of file

**cp** - used to copy files or directories

**mv** - used to move or rename a directory

**grep** - search through text for a pattern

**who** - show all logged in users

  - **delivers a SIGINT signal to the program, used for killing a program.**

# Linux Command Example

```
aj654@ilab1:~$
```

# Linux Command Example

```
aj654@ilab1:~$ ls
```

# Linux Command Example

```
aj654@ilab1:~$ ls  
someDirectory  
aj654@ilab1:~$
```

# Linux Command Example

```
aj654@ilab1:~$ ls  
someDirectory  
aj654@ilab1:~$ cd someDirectory
```

# Linux Command Example

```
aj654@ilab1:~$ ls
someDirectory
aj654@ilab1:~$ cd someDirectory
aj654@ilab1:~/someDirectory$
```

# Linux Command Example

```
aj654@ilab1:~$ ls
someDirectory
aj654@ilab1:~$ cd someDirectory
aj654@ilab1:~/someDirectory$ touch file1
```



# Linux Command Example

```
aj654@ilab1:~$ ls
someDirectory
aj654@ilab1:~$ cd someDirectory
aj654@ilab1:~/someDirectory$ touch file1
aj654@ilab1:~/someDirectory$
```

# Linux Command Example

```
aj654@ilab1:~$ ls
someDirectory
aj654@ilab1:~$ cd someDirectory
aj654@ilab1:~/someDirectory$ touch file1
aj654@ilab1:~/someDirectory$ ls
```

# Linux Command Example

```
aj654@ilab1:~$ ls
someDirectory
aj654@ilab1:~$ cd someDirectory
aj654@ilab1:~/someDirectory$ touch file1
aj654@ilab1:~/someDirectory$ ls
file1
aj654@ilab1:~/someDirectory$
```

# Linux Command Example

```
aj654@ilab1:~$ ls
someDirectory
aj654@ilab1:~$ cd someDirectory
aj654@ilab1:~/someDirectory$ touch file1
aj654@ilab1:~/someDirectory$ ls
file1
aj654@ilab1:~/someDirectory$ mv file1 newName
```

# Linux Command Example

```
aj654@ilab1:~$ ls
someDirectory
aj654@ilab1:~$ cd someDirectory
aj654@ilab1:~/someDirectory$ touch file1
aj654@ilab1:~/someDirectory$ ls
file1
aj654@ilab1:~/someDirectory$ mv file1 newName
aj654@ilab1:~/someDirectory$
```

# Linux Command Example

```
aj654@ilab1:~$ ls
someDirectory
aj654@ilab1:~$ cd someDirectory
aj654@ilab1:~/someDirectory$ touch file1
aj654@ilab1:~/someDirectory$ ls
file1
aj654@ilab1:~/someDirectory$ mv file1 newName
aj654@ilab1:~/someDirectory$ ls
```

# Linux Command Example

```
aj654@ilab1:~$ ls
someDirectory
aj654@ilab1:~$ cd someDirectory
aj654@ilab1:~/someDirectory$ touch file1
aj654@ilab1:~/someDirectory$ ls
file1
aj654@ilab1:~/someDirectory$ mv file1 newName
aj654@ilab1:~/someDirectory$ ls
newName
aj654@ilab1:~/someDirectory$
```

# Linux Command Example

```
aj654@ilab1:~$ ls
someDirectory
aj654@ilab1:~$ cd someDirectory
aj654@ilab1:~/someDirectory$ touch file1
aj654@ilab1:~/someDirectory$ ls
file1
aj654@ilab1:~/someDirectory$ mv file1 newName
aj654@ilab1:~/someDirectory$ ls
newName
aj654@ilab1:~/someDirectory$ cd ..
```



# Linux Command Example

```
aj654@ilab1:~$ ls
someDirectory
aj654@ilab1:~$ cd someDirectory
aj654@ilab1:~/someDirectory$ touch file1
aj654@ilab1:~/someDirectory$ ls
file1
aj654@ilab1:~/someDirectory$ mv file1 newName
aj654@ilab1:~/someDirectory$ ls
newName
aj654@ilab1:~/someDirectory$ cd ..
aj654@ilab1:~$
```

# The C Programming Language

- **Procedural** (as opposed to object-oriented).
- **Statically** typed.
  - considered both **strongly** and **weakly** typed.
- C provides control over a Von Neumann machines abstractions.
  - We will use the C17 standard (ISO/IEC 9899:2018) in this course.

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int num = 10;
    float pi = 3.14;
    char letter = 'A';

    printf("Integer: %d\n", num);
    printf("Float: %f\n", pi);
    printf("Character: %c\n", letter);

    return EXIT_SUCCESS;
}
```

# Data Types in C

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    int num = 10;
    float pi = 3.14;
    char letter = 'A';

    printf("Integer: %d\n", num);
    printf("Float: %f\n", pi);
    printf("Character: %c\n", letter);

    return EXIT_SUCCESS;
}
```

# Functions in C

```
#include <stdio.h>
#include <stdlib.h>
int add(int a, int b) {
    return a + b;
}

int main(void) {
    int result = add(3, 5);
    printf("Sum: %d\n", result);

    return EXIT_SUCCESS;
}
```

# Conditional in C

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    int num = 7;

    if (num % 2 == 0) {
        printf("Even\n");
    } else {
        printf("Odd\n");
    }

    return EXIT_SUCCESS;
}
```

# Loops in C

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    for (int i = 1; i <= 5; i++) {
        printf("%d ", i);
    }
    return EXIT_SUCCESS;
}
```

# Structs in C

```
#include <stdio.h>
#include <stdlib.h>
struct Point {
    int x;
    int y;
};

int main(void) {
    struct Point p1 = {3, 5};
    printf("Coordinates: (%d, %d)\n", p1.x, p1.y);

    return EXIT_SUCCESS;
}
```

# Compiling a C program

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    for(int i = 1; i <= 5; i++) {
        printf("%d ", i);
    }
    printf("\n");
    return EXIT_SUCCESS;
}
```

```
aj654@ilab1:~$ ls
example.c
aj654@ilab1:~$ gcc -Wall -Werror
-fsanitize=address,undefined example.c
-o example
aj654@ilab1:~/someDirectory$ ls
example.c example
aj654@ilab1:~/someDirectory$ ./example
1 2 3 4 5
aj654@ilab1:~/someDirectory$
```



# What is Make?

- Make is a build automation tool that reads a *makefile* and executes the specified commands.
  - Automatically rebuilds components based on file timestamp.
- We will use Make to automate the process of compiling C programs.



**GNU Make**

# The Anatomy of a *makefile*

```
aj654@ilab1:~$ ls  
makefile
```

```
all: compile
```

```
compile: example.c
```

```
    gcc -Wall -Werror -fsanitize=address,undefined example.c -o example
```

```
clean:
```

```
    rm -rf example
```

# The Anatomy of a *makefile*

```
aj654@ilab1:~$ ls  
makefile
```

## *Rule*

```
all: compile
```

```
compile: example.c
```

```
    gcc -Wall -Werror -fsanitize=address,undefined example.c -o example
```

```
clean:
```

```
    rm -rf example
```

# The Anatomy of a *makefile*

```
aj654@ilab1:~$ ls  
makefile
```

## *Rule Target*

```
all: compile
```

```
compile: example.c
```

```
gcc -Wall -Werror -fsanitize=address,undefined example.c -o example
```

```
clean:
```

```
rm -rf example
```

# The Anatomy of a *makefile*

```
aj654@ilab1:~$ ls  
makefile
```

## Rule Target Dependency

```
all: compile
```

```
compile: example.c
```

```
gcc -Wall -Werror -fsanitize=address,undefined example.c -o example
```

```
clean:
```

```
rm -rf example
```

# The Anatomy of a *makefile*

```
aj654@ilab1:~$ ls  
makefile
```

## Rule Target Dependency Command

```
all: compile
```

```
compile: example.c
```

```
gcc -Wall -Werror -fsanitize=address,undefined example.c -o example
```

```
clean:
```

```
rm -rf example
```

# The Anatomy of a *makefile*

```
aj654@ilab1:~$ ls  
makefile
```

## Rule Target Dependency Command

```
all: compile
```

```
compile: example.c
```

```
gcc -Wall -Werror -fsanitize=address,undefined example.c -o example
```

```
clean:
```

```
rm -rf example
```

**Note: a single tab, NOT 4 spaces!!**

# A more advanced *makefile*

```
CC=gcc
CFLAGS=-Wall -Werror -fsanitize=address,undefined
MAIN=example

all: $(MAIN)

.SECONDEXPANSION:
$(MAIN): $$@.c
    $(CC) $(CFLAGS) $^ -o $@

clean:
    rm -rf $(MAIN)
```



# A more advanced *makefile*

## *Variables*

```
CC=gcc
CFLAGS=-Wall -Werror -fsanitize=address,undefined
MAIN=example
```

```
all: $(MAIN)
```

```
.SECONDEXPANSION:
```

```
$(MAIN): $$@.c
```

```
$(CC) $(CFLAGS) $^ -o $@
```

```
clean:
```

```
rm -rf $(MAIN)
```

# A more advanced *makefile*

```
CC=gcc
CFLAGS=-Wall -Werror -fsanitize=address,undefined
MAIN=example
```

```
all: $(MAIN)
```

```
.SECONDEXPANSION:
```

```
$(MAIN): $$@.c
    $(CC) $(CFLAGS) $^ -o $@
```

```
clean:
```

```
rm -rf $(MAIN)
```

**Variables**  
**Automatic Variables**

# A more advanced *makefile*

```
CC=gcc
CFLAGS=-Wall -Werror -fsanitize=address,undefined
MAIN=example
```

```
all: $(MAIN)
```

```
.SECONDEXPANSION:
```

```
$(MAIN): $$@.c
```

```
$(CC) $(CFLAGS) $^ -o $@
```

```
clean:
```

```
rm -rf $(MAIN)
```

*Variables*  
*Automatic Variables*  
*Special Target*

# A program to sum prime numbers up to $n$

- Idea: iterate through all numbers up to  $n$ 
  - If the number is prime, add to accumulator

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

bool is_prime(long long number)
{
    for(long long i = 2; i < number; i++)
    {
        if(number % i == 0 && i != number) return false;
    }
    return true;
}

long long sum_primes(long long limit)
{
    long long sum = 0;
    for(long long i = 2; i <= limit; i++)
    {
        if(is_prime(i)) sum+=i;
    }
    return sum;
}

int main(int argc, char *argv[])
{
    char *p;
    printf("%lld\n", sum_primes(strtoll(argv[1], &p, 10)));
    return EXIT_SUCCESS;
}
```

# A program to sum prime numbers up to $n$

- Idea: iterate through all numbers up to  $n$ 
  - If the number is prime, add to accumulator
- Slow! If 4 is not prime, then of course 8 is also not prime
  - Can we exploit this observation to speed up the computation?

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

bool is_prime(long long number)
{
    for(long long i = 2; i < number; i++)
    {
        if(number % i == 0 && i != number) return false;
    }
    return true;
}

long long sum_primes(long long limit)
{
    long long sum = 0;
    for(long long i = 2; i <= limit; i++)
    {
        if(is_prime(i)) sum+=i;
    }
    return sum;
}

int main(int argc, char *argv[])
{
    char *p;
    printf("%lld\n", sum_primes(strtoll(argv[1], &p, 10)));
    return EXIT_SUCCESS;
}
```

# Sieve of Eratosthenes (eh-ruh-taas-thuh-nee-z)

- Create a list of numbers 2 to  $n$ .
- Mark multiples of each number up to  $n$ .
  - The unmarked numbers are the primes.
- We will use variable-length arrays (C99+, but bad to use), next recitation we will cover dynamic arrays.

|     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 2   | 3   | 5   | 7   |
| 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19  | 20  | 11  | 13  | 17  | 19  |
| 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 23  | 29  |     |     |
| 31  | 32  | 33  | 34  | 35  | 36  | 37  | 38  | 39  | 40  | 31  | 37  |     |     |
| 41  | 42  | 43  | 44  | 45  | 46  | 47  | 48  | 49  | 50  | 41  | 43  | 47  |     |
| 51  | 52  | 53  | 54  | 55  | 56  | 57  | 58  | 59  | 60  | 53  | 59  |     |     |
| 61  | 62  | 63  | 64  | 65  | 66  | 67  | 68  | 69  | 70  | 61  | 67  |     |     |
| 71  | 72  | 73  | 74  | 75  | 76  | 77  | 78  | 79  | 80  | 71  | 73  | 79  |     |
| 81  | 82  | 83  | 84  | 85  | 86  | 87  | 88  | 89  | 90  | 83  | 89  |     |     |
| 91  | 92  | 93  | 94  | 95  | 96  | 97  | 98  | 99  | 100 | 97  |     |     |     |
| 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 101 | 103 | 107 | 109 |
| 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 113 |     |     |     |

# Sieve of Eratosthenes (eh-ruh-taas-thuh-nee-z)

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

long long sum_primes(long long limit);

int main(int argc, char *argv[]){
    char *p;
    printf("%lld\n", sum_primes(strtoll(argv[1], &p, 10)));

    return EXIT_SUCCESS;
}
```

```
long long sum_primes(long long limit){
    long long sum = 0;
    bool marks[limit+1];

    for(long long i=0; i < limit+1; i++){
        marks[i] = false;
    }

    for(long long i = 2; i <= limit; i++){

        if(!marks[i]){
            sum+=i;

            for(long long m=2*i; m <= limit; m += i)
            {
                marks[m] = true;
            }
        }
    }
    return sum;
}
```

# Performance Comparison

| Limit  | Naive Runtime (S) | Sieve Runtime (S) |
|--------|-------------------|-------------------|
| 100000 | 0.87              | 0.004             |
| 150000 | 1.893             | 0.005             |
| 200000 | 3.26              | 0.005             |
| 250000 | 5.092             | 0.005             |
| 300000 | 7.127             | 0.006             |
| 350000 | 9.624             | 0.006             |
| 400000 | 12.46             | 0.006             |
| 450000 | 15.569            | 0.007             |
| 500000 | 18.993            | 0.007             |