Recitation 1

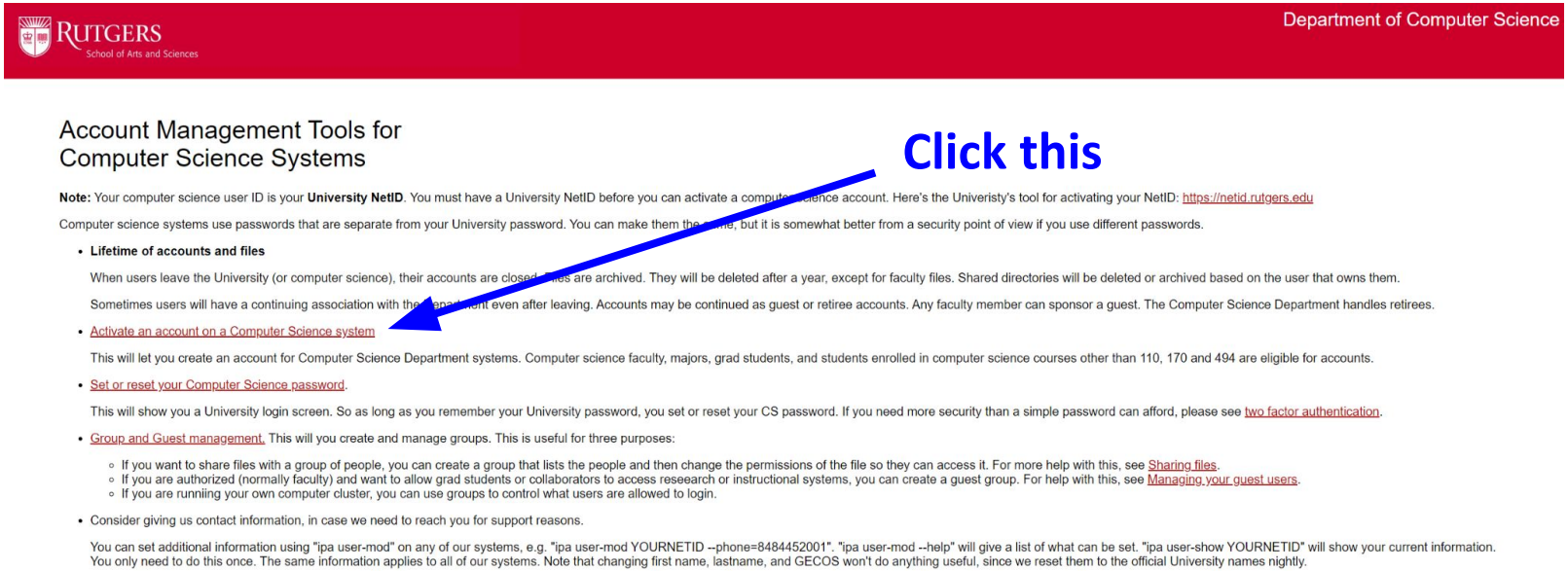
Computer Architecture (section 1)

About Me

- Name: Alborz Jelvani
- Email: 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

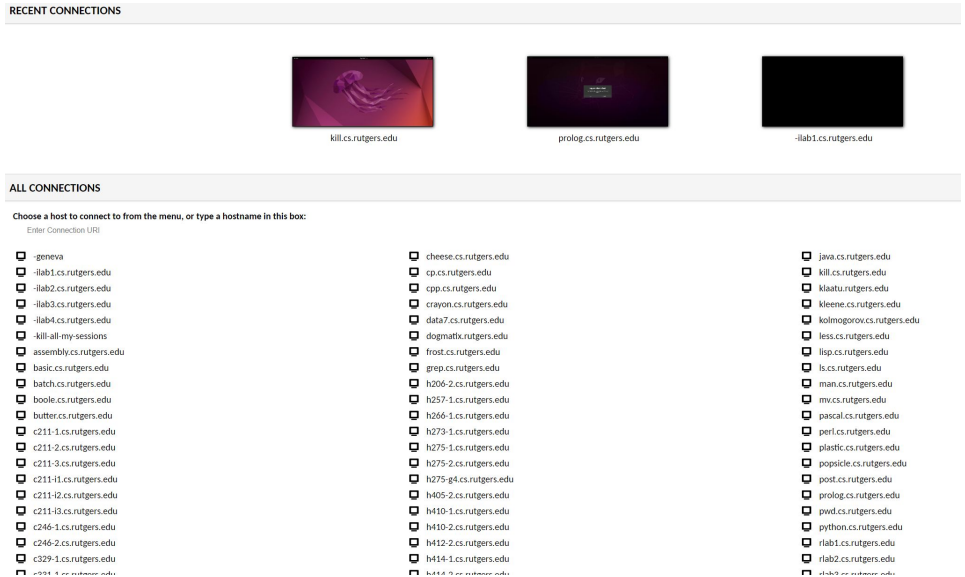
- https://services.cs.rutgers.edu/accounts/



Weblogin for Rutgers CS

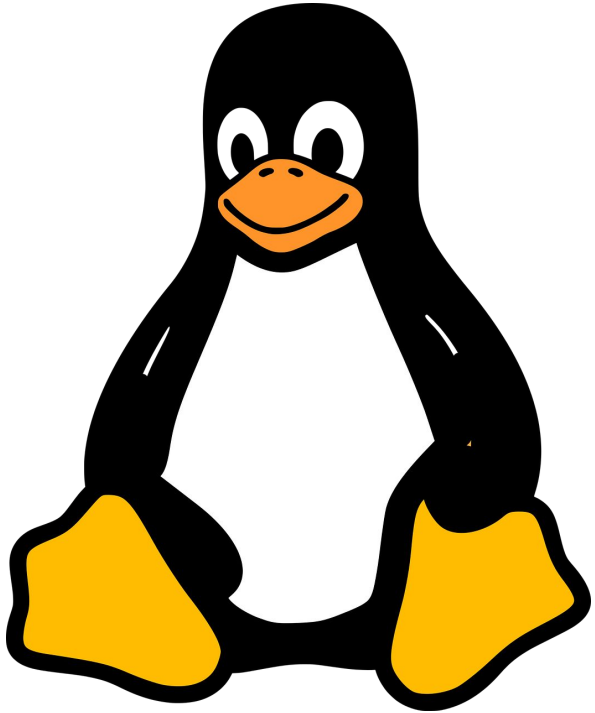
- https://weblogin.cs.rutgers.edu/
  - Login to Rutgers CS account
  - Pick a machine

- https://report.cs.rutgers.edu/nagiosnotes/iLab-machines.html
  - View machine status



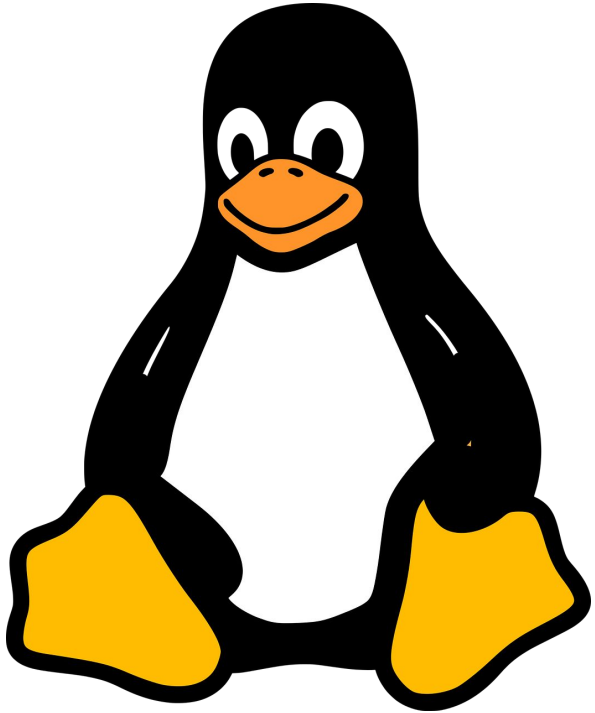
Linux Shell Intro

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



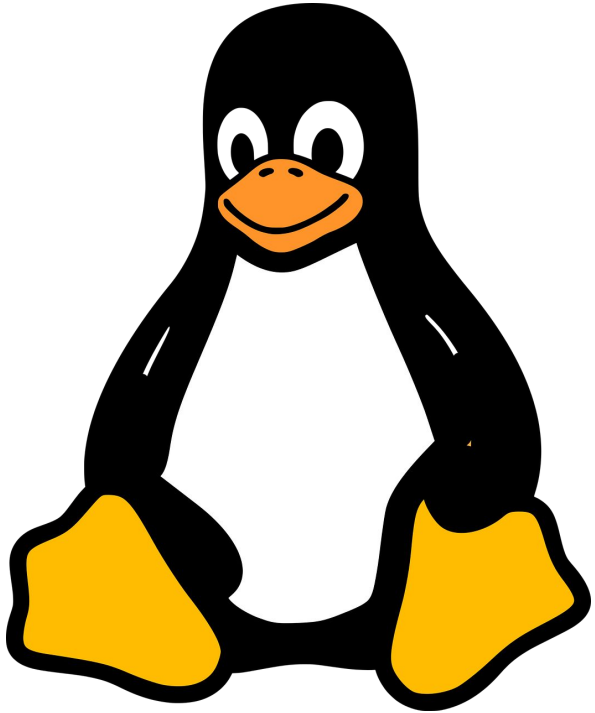
Linux Shell Intro

- 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.**

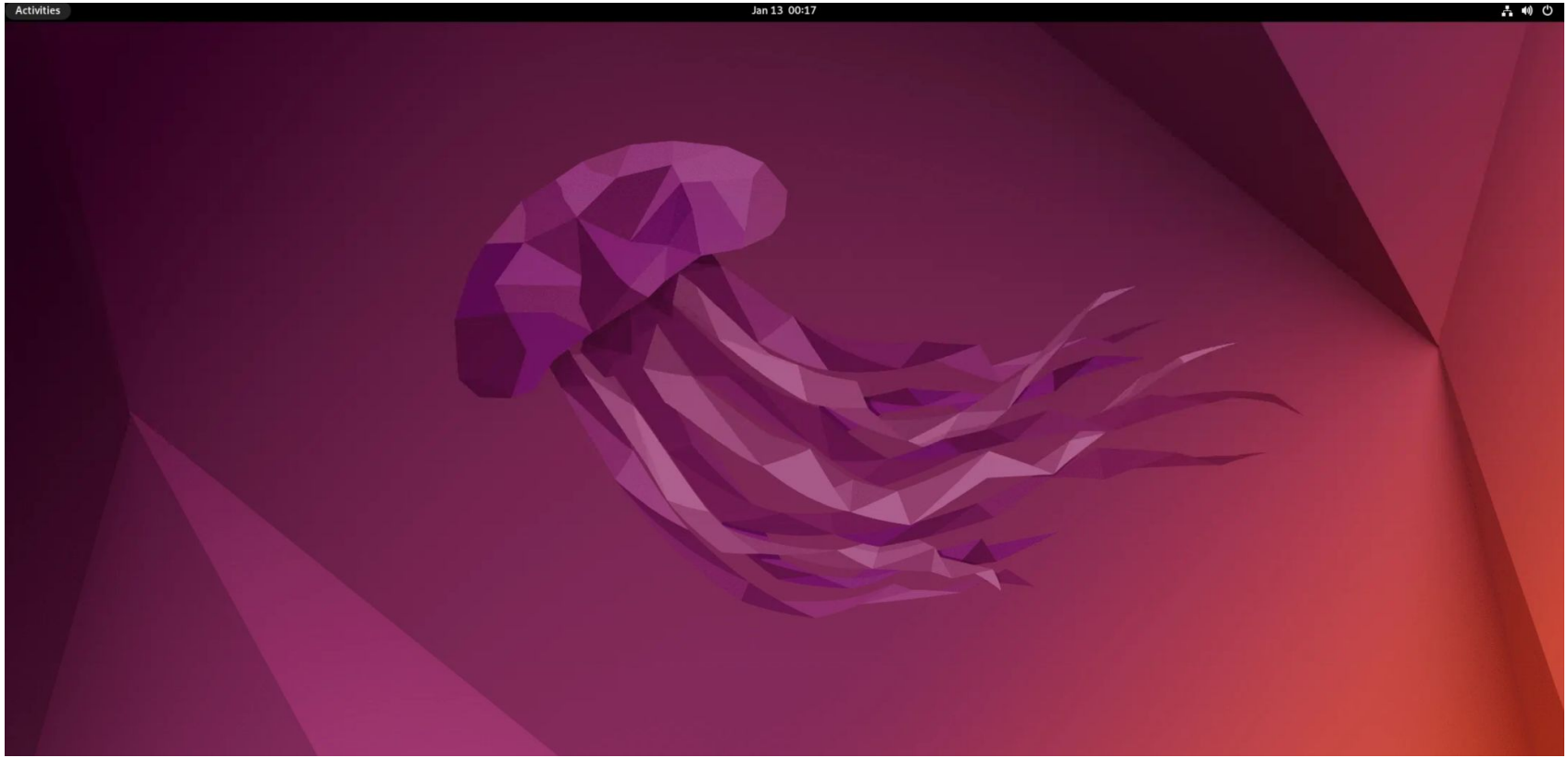


Linux Shell Intro

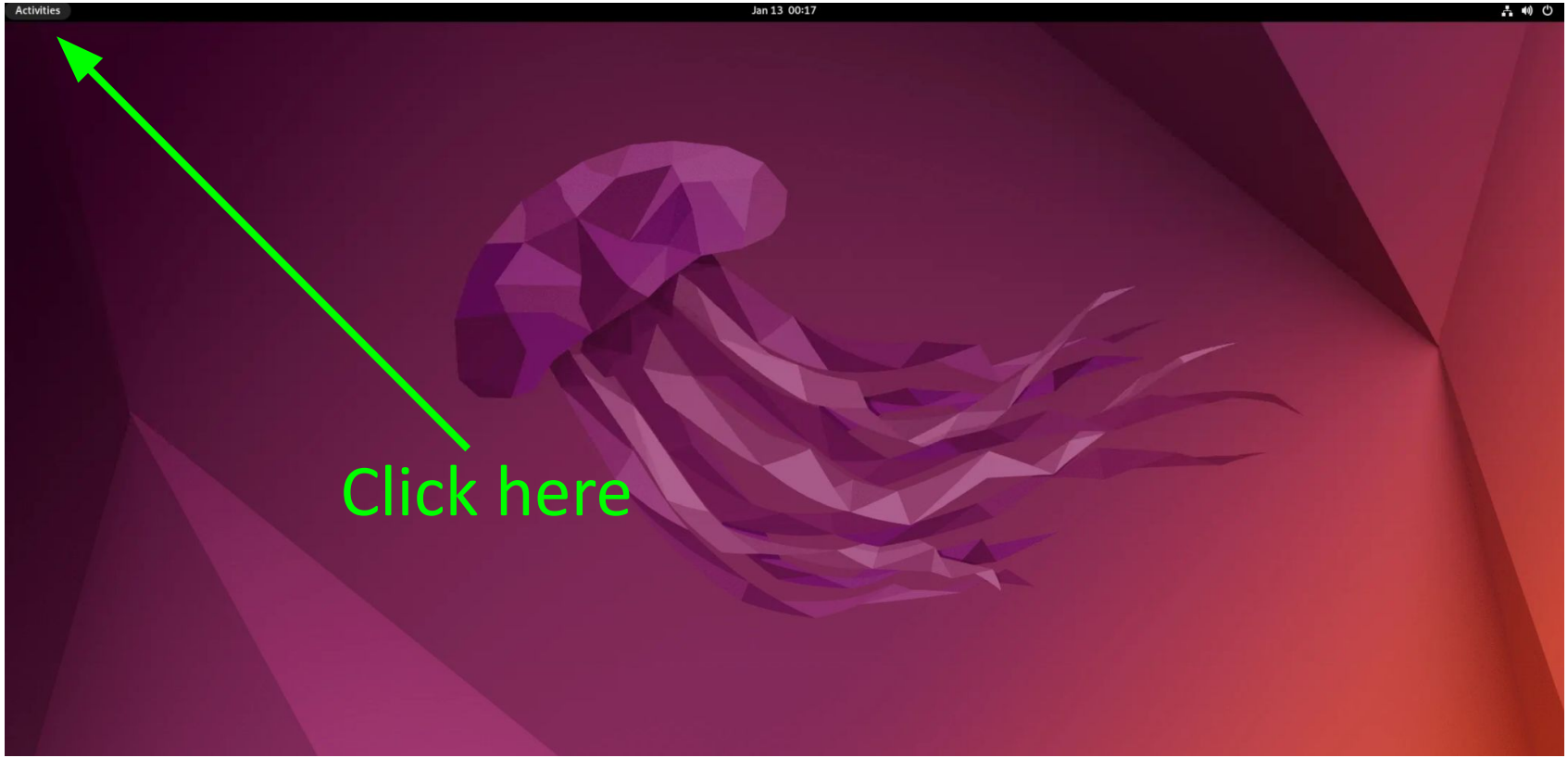
- Interaction with the OS occurs through the ***shell.***
- The ***shell*** interprets commands and prints responses.
  - The default shell in Linux is a program called **bash.**
- A ***terminal emulator*** (AKA terminal) is a program that provides an interface between the user and a shell.



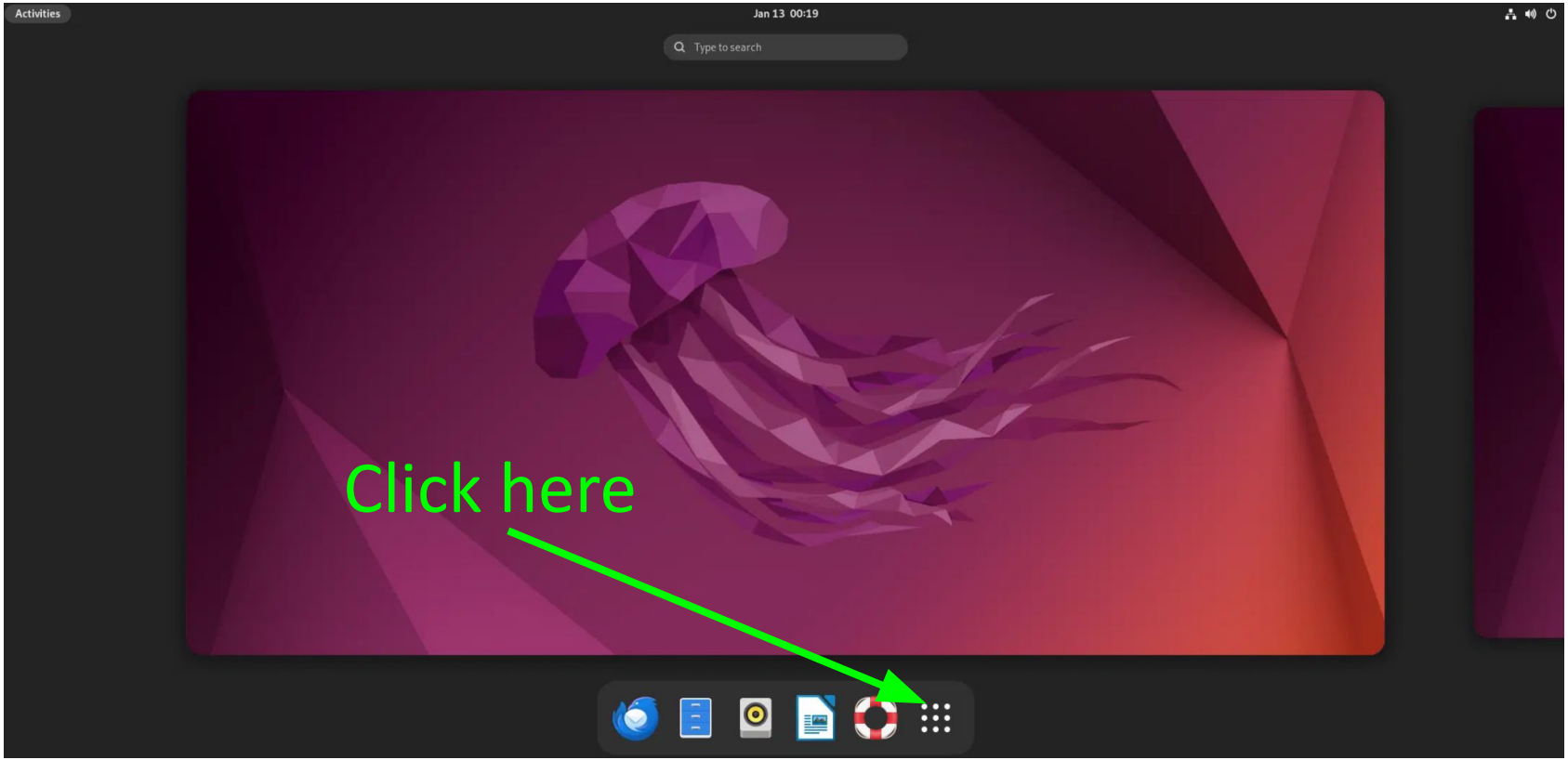
Opening a Terminal on iLab



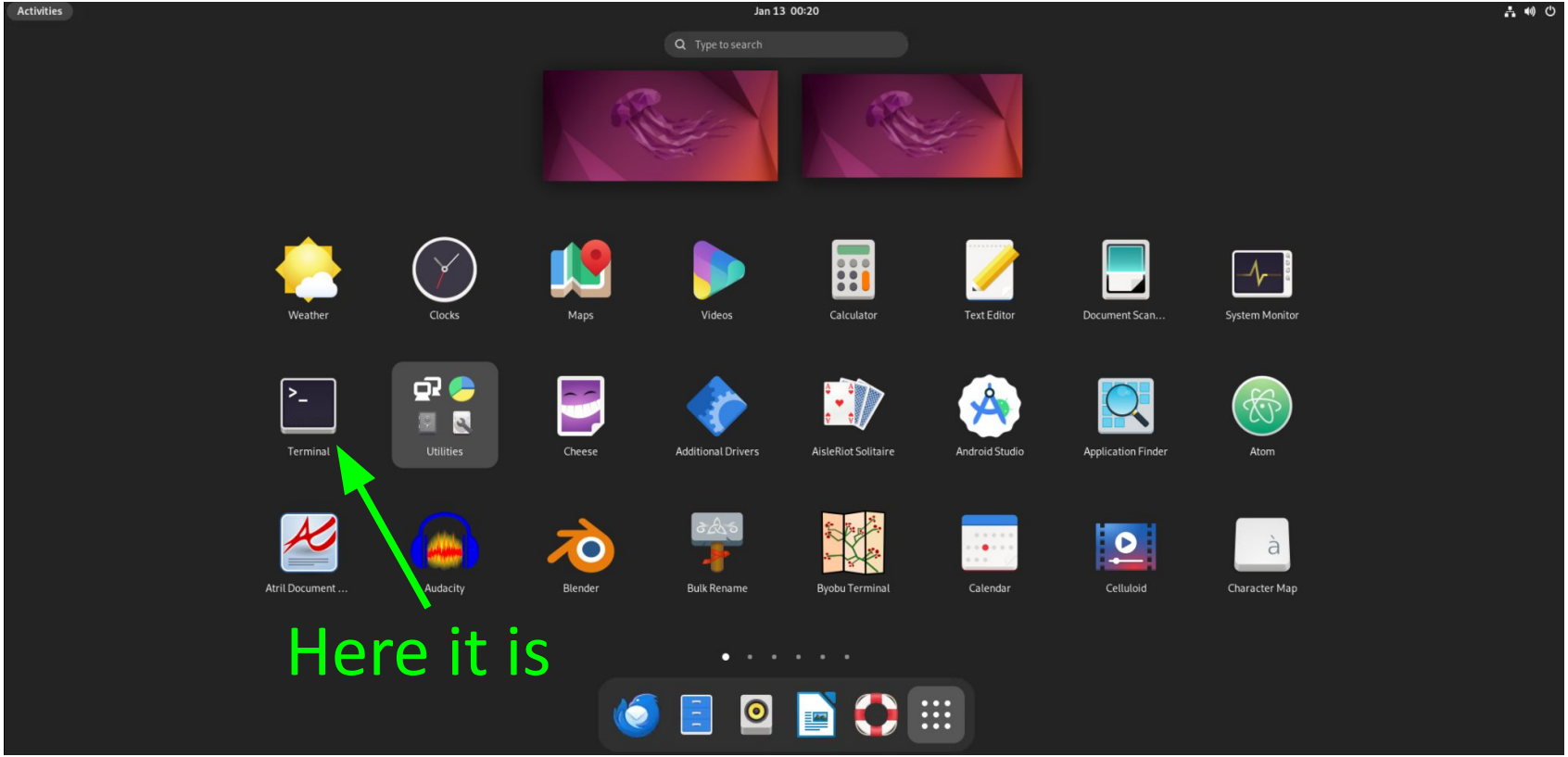
Opening a Terminal on iLab



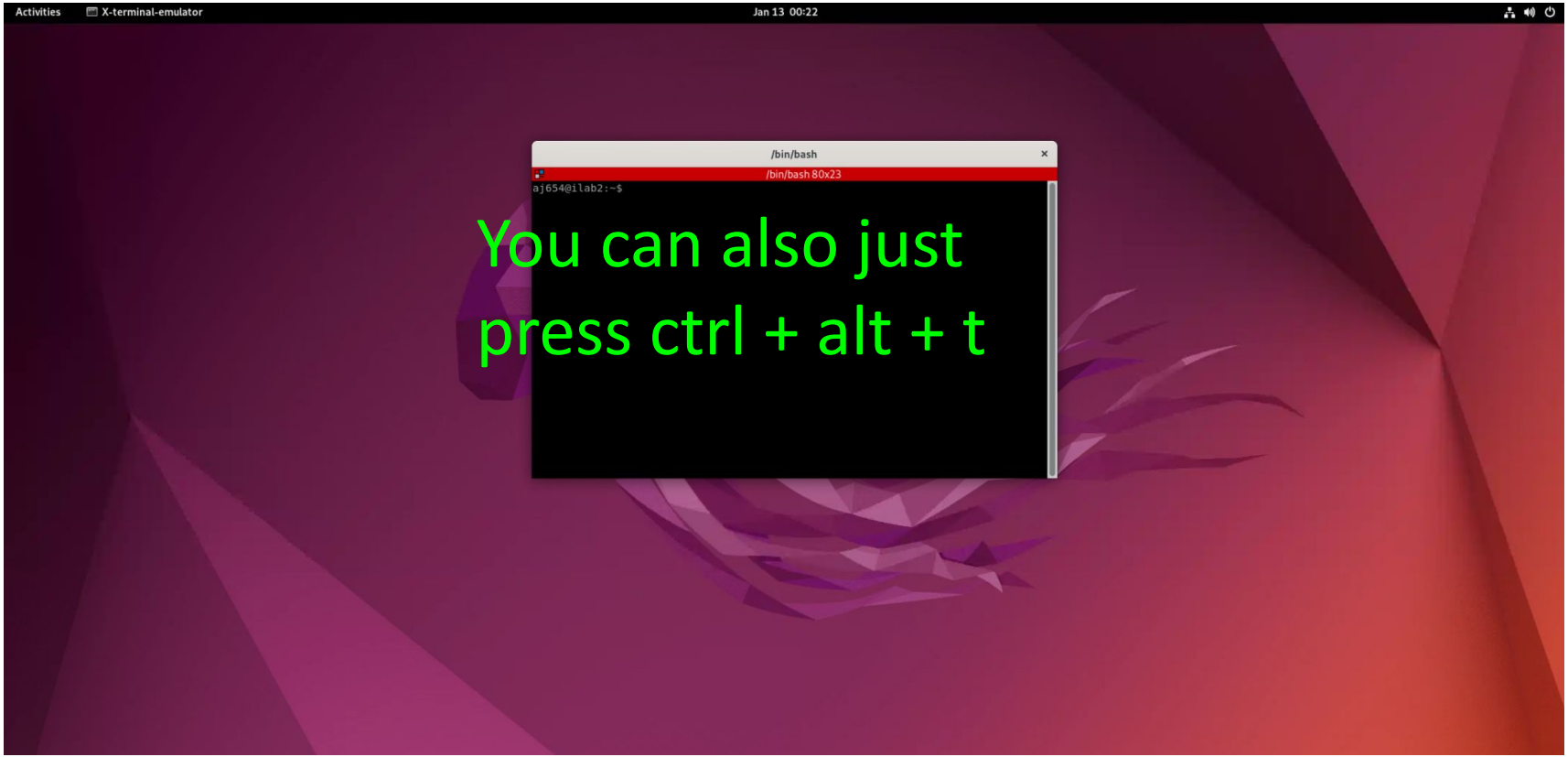
Opening a Terminal in Ubuntu



Opening a Terminal in Ubuntu

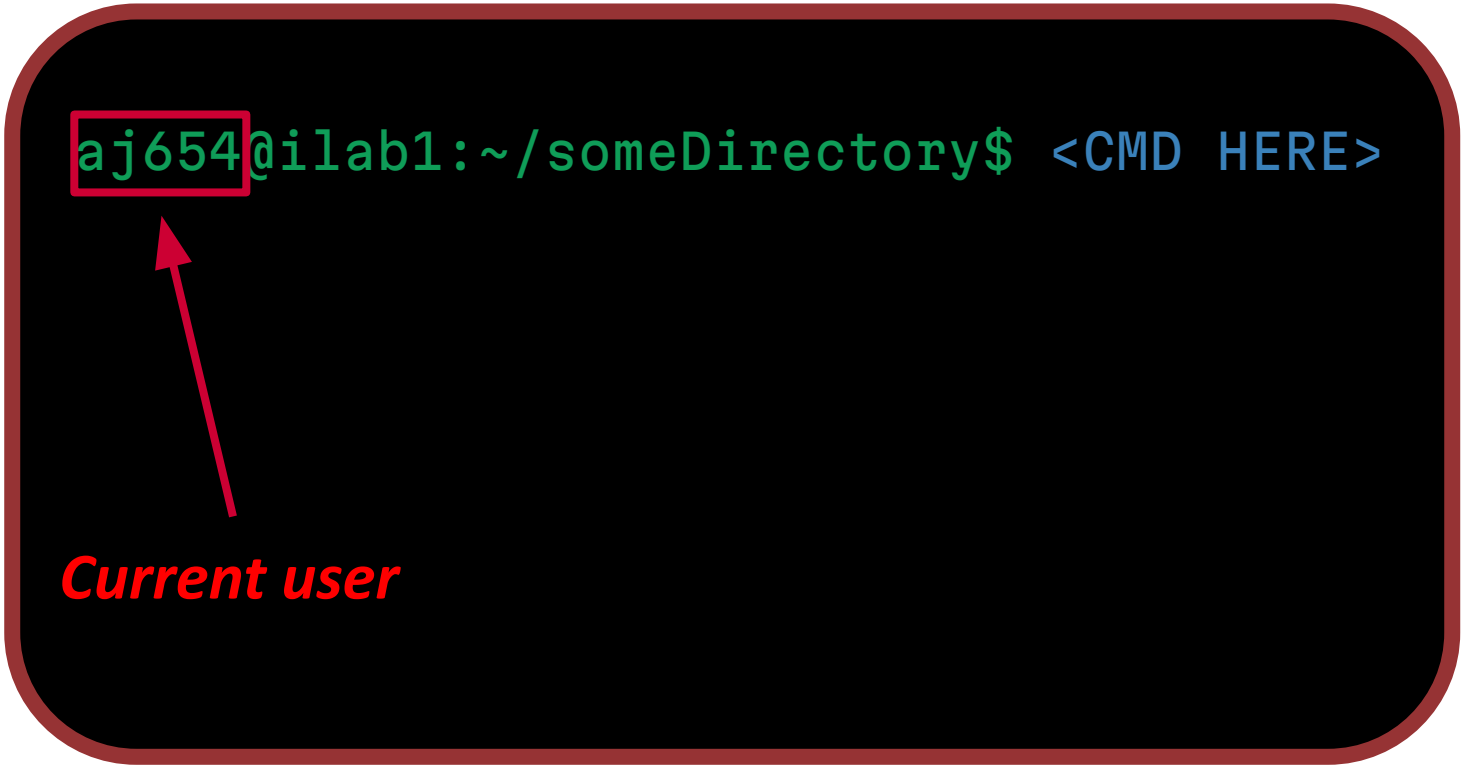


Opening a Terminal on iLab

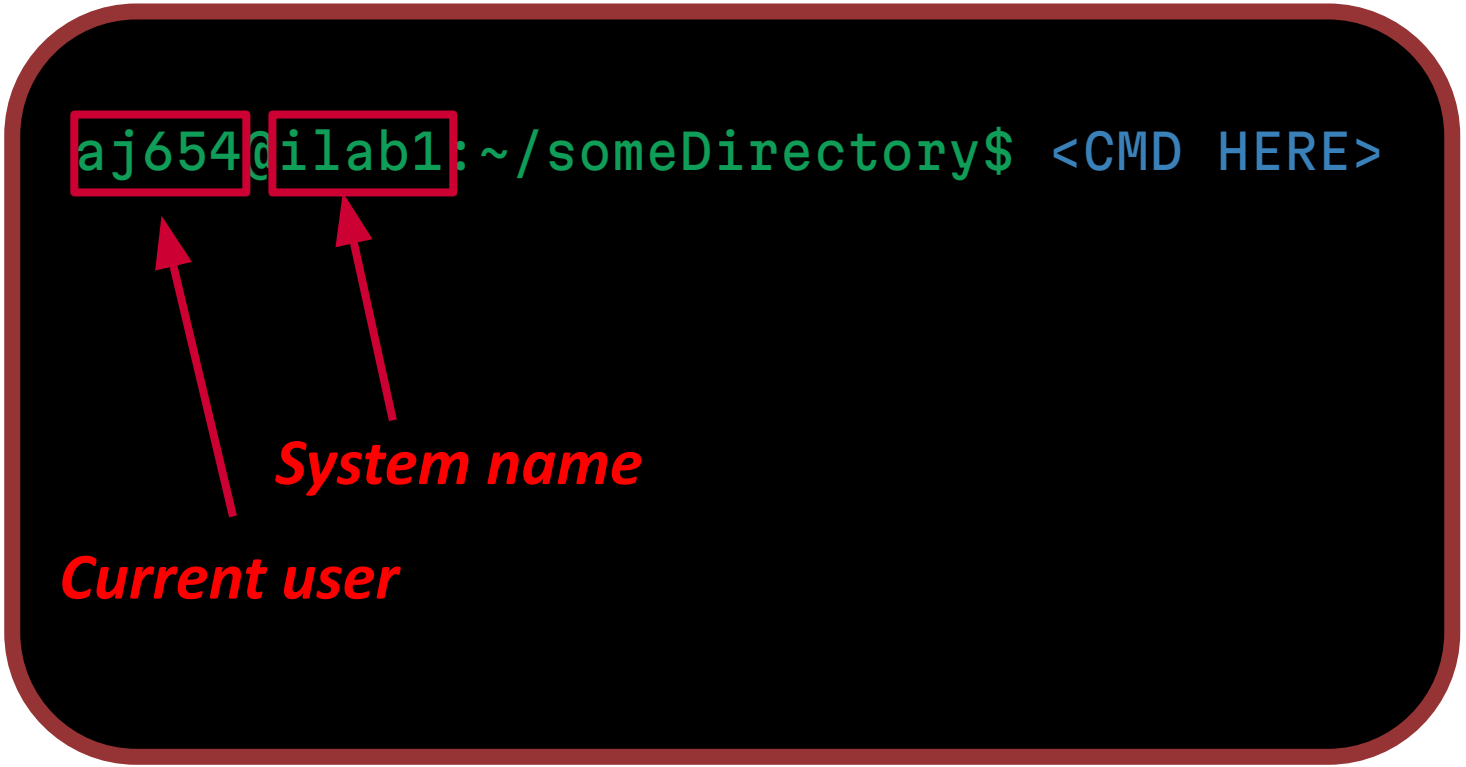


Linux Shell Anatomy

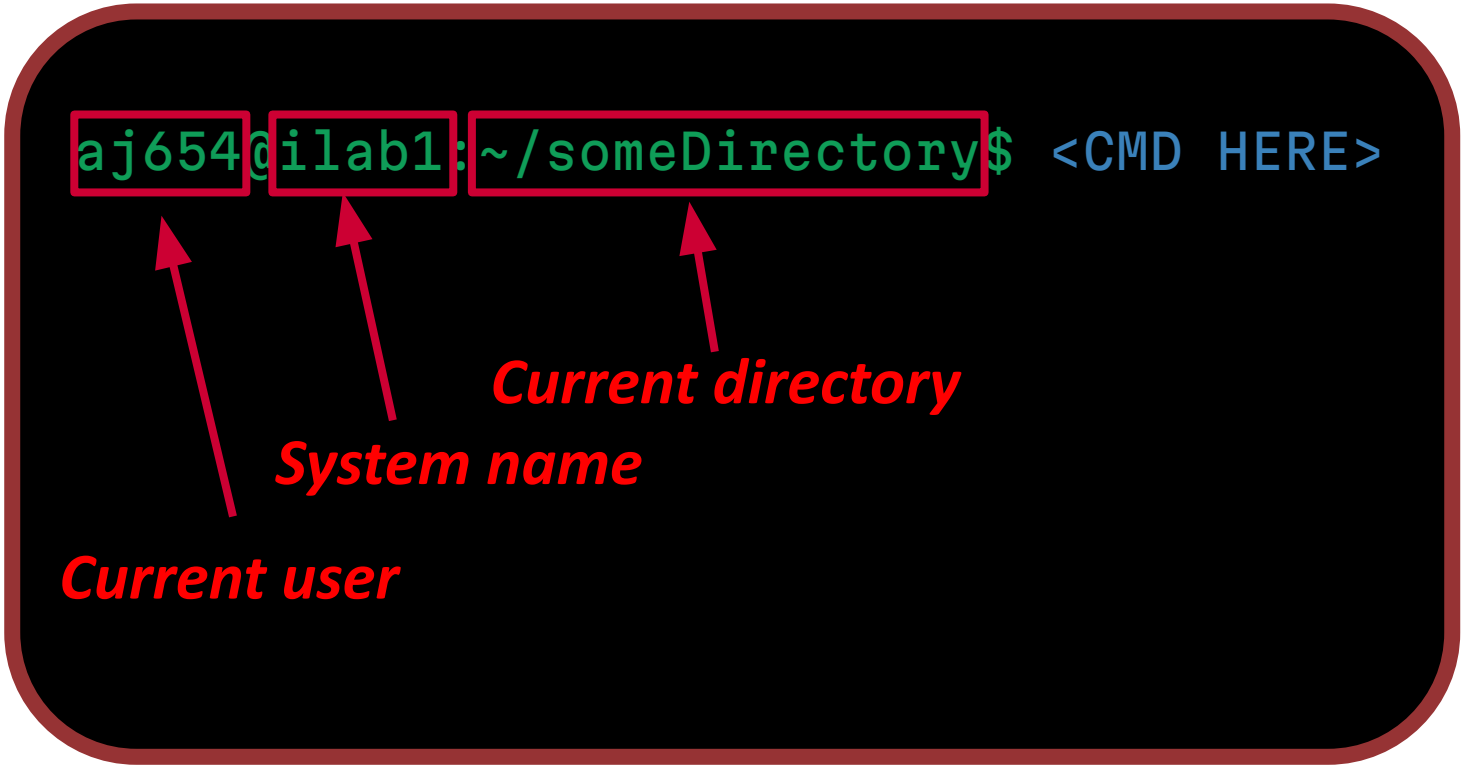
Linux Shell Anatomy



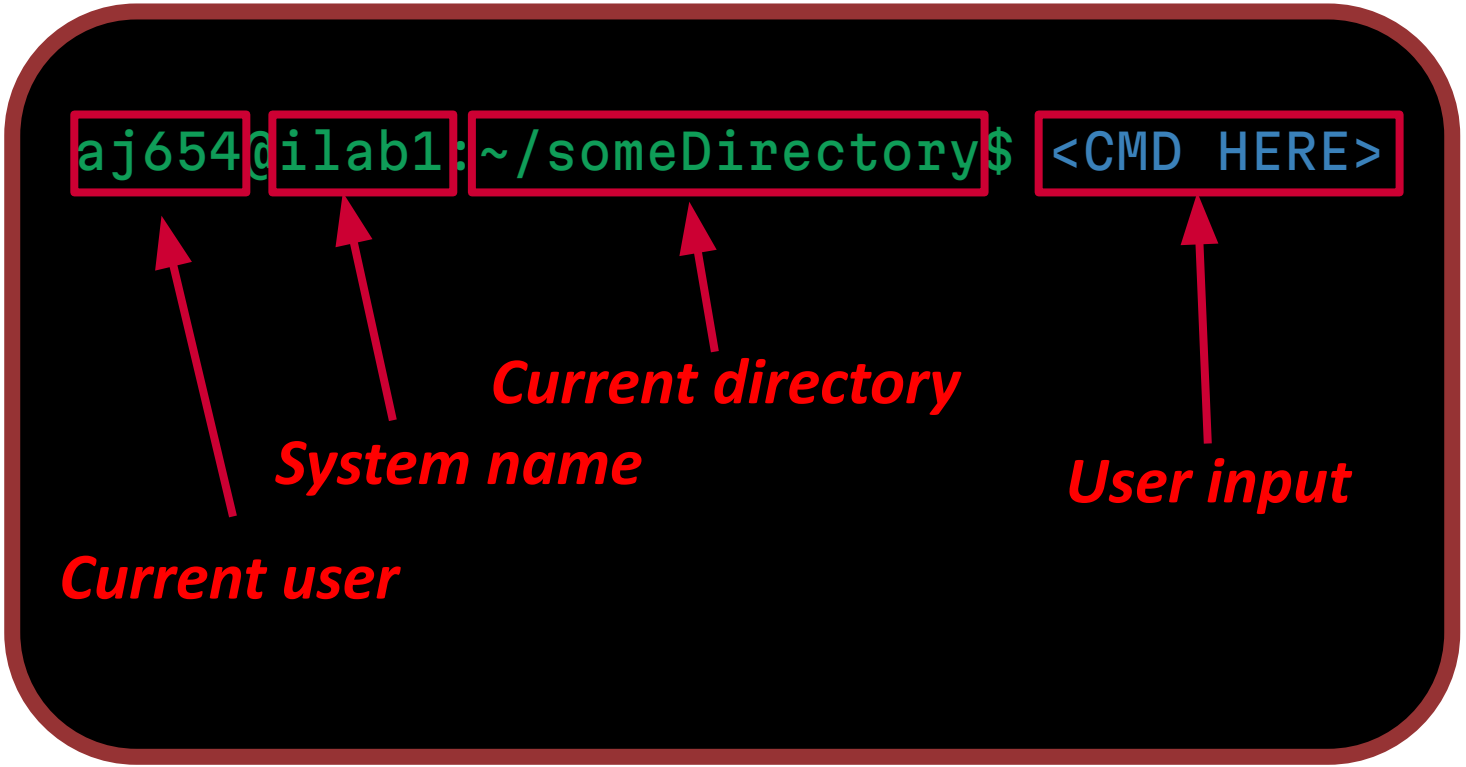
Linux Shell Anatomy



Linux Shell Anatomy



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

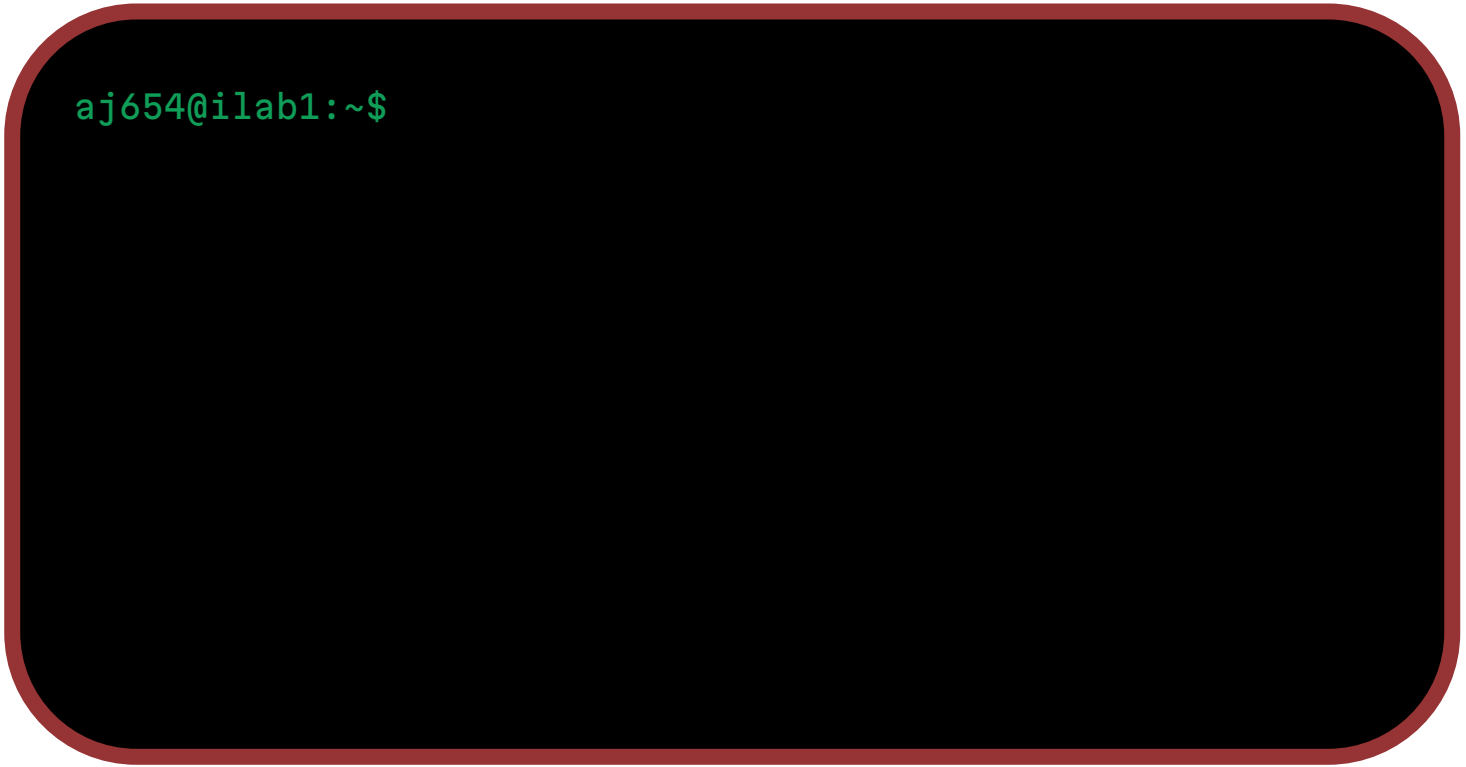
mv - used to move or rename a directory

grep - search through text for a pattern

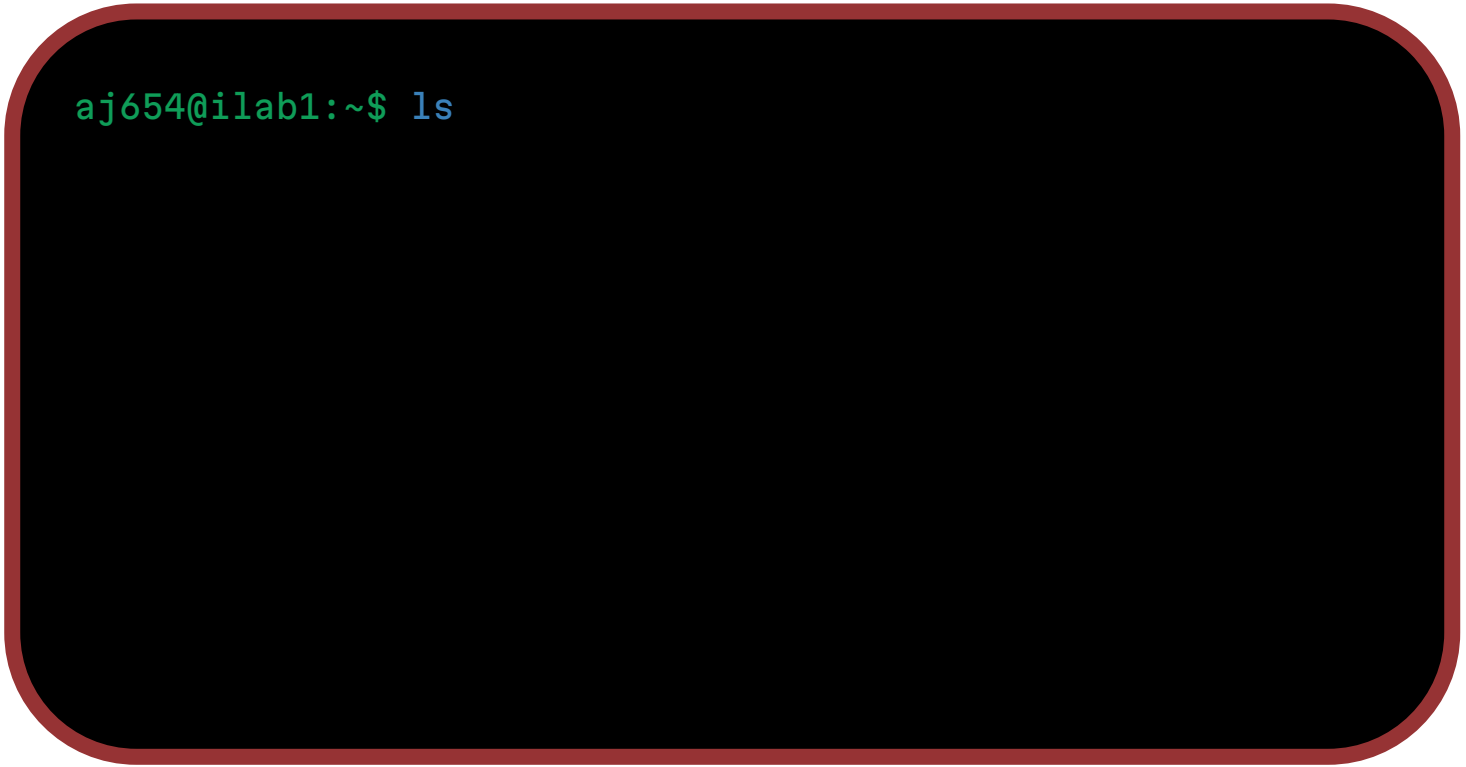
who - show all logged in users

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

Linux Command Example



Linux Command Example



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.

```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;
}
```

Data Types in C

```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

```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

```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

```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

```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

```c
#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*

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

all: $(MAIN)

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

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

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

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

```c
#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;
}
```

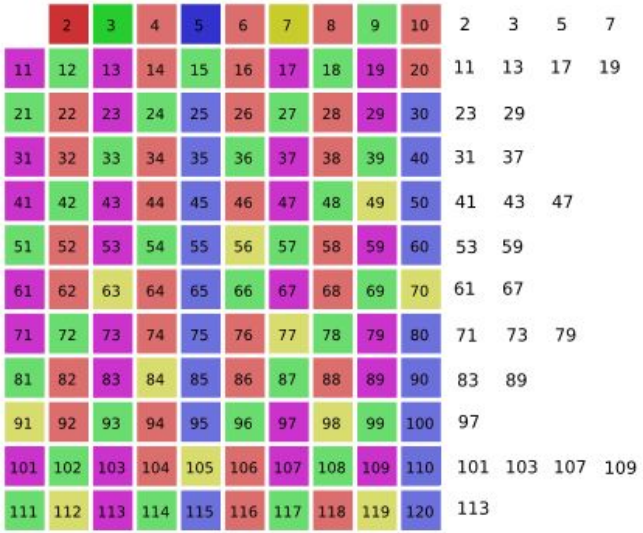
A program to sum prime numbers up to *n*

- 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-neez)

- 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.



Sieve of Eratosthenes (eh-ruh-taas-thuh-neez)

```c
#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;
}
```

```c
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