MODULE 03 – HIGH PERFORMANCE COMPUTING

UNIX 101

HYUN MIN KANG @ UNIVERSITY OF MICHIGAN

Today, I hope that you will...

- Agree that `UNIX` is a useful platform to perform (big) data science for academic research.
- Understand the high-level picture and philosophy of `UNIX`
- Have a basic understanding of how to build a modular workflow in `UNIX` platform by utilizing existing tools.
- Practice building modular workflow with simple examples.

Motivation

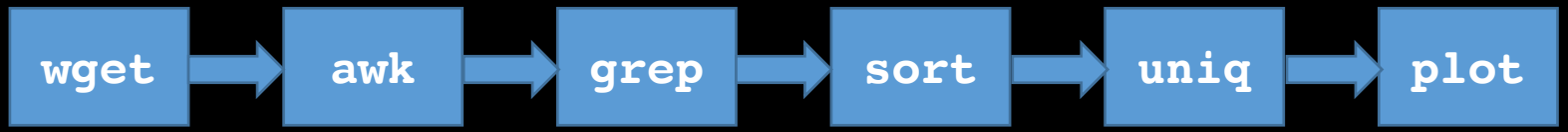
Why do we use UNIX?

Modularity	**Core programs are modular and work well with others**
Programmability	**Arguably the best existing software development environment**
Infrastructure	**Access to existing tools and cutting-edge methods**
Reliability	**Unparalleled uptime and stability**
Unix Philosophy	**Encourages open standards**

Modularity	Core programs are modular and work well with others
Programmability	Arguably the best existing software development environment
Infrastructure	Access to existing tools and cutting-edge methods
Reliability	Unparalleled uptime and stability
Unix Philosophy	Encourages open standards

Modularity

- **UNIX** shell allows users to build a complex workflow by interfacing smaller modular programs together



- Alternatively, one could write a single complex program.

Which one do you prefer and why?

Advantages and disadvantages

- **“Monster approach” is..**
  - Customized to a particular project
    - For frequent tasks, customized optimization and interface can be useful.
  - Resulting code can be massive, fragile, inflexible, and untransferrable.

- **With modular workflow it is easier to**
  - Utilize existing modules without having to implement new code
  - Spot errors and figure out where they’re occurring by inspecting intermediate results
  - Experiment with alternative methods by swapping out components
  - Tackle novel problems by remixing existing modular tools

Monster Approach vs. Modular Workflow

UNIX Philosophy

“Write programs that do one thing and do it well. Write programs to work together and that encourage open standards.
Write programs to handle text streams, because that is a universal interface.”

— Doug McIlory

Basics	File Control	View/Edit	Misc.	Power Commands	Process Related
ls	mv	less	chmod	grep	top
cd	cp	head	cat	cut	ps
pwd	mkdir	tail	tee	sort	kill
man	rm	nano	wget	uniq	Ctrl-c
apropos	| (pipe)	touch	wc	xargs	Ctrl-z
ssh	> (write to file)	od	echo	find	bg
export	< (read from file)		zcat	sed	fg
	scp		source	awk	screen

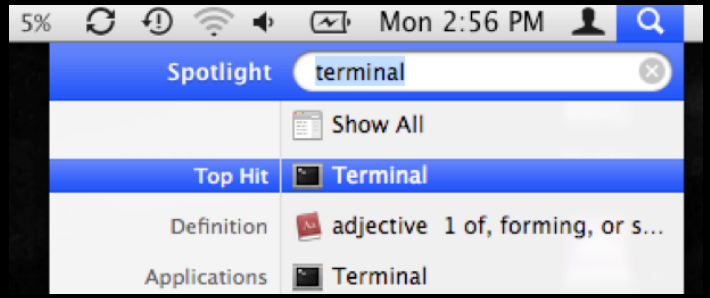
Basics	File Control	View/Edit	Misc.	Power Commands	Process Related
ls	mv	less	chmod	grep	top
cd	cp	head	cat	cut	ps
pwd	mkdir	tail	tee	sort	kill
man	rm	nano	wget	uniq	Ctrl-c
apropos	| (pipe)	touch	wc	xargs	Ctrl-z
ssh	> (write to file)	od	echo	find	bg
export	< (read from file)		zcat	sed	fg
	scp		source	awk	screen

Meeting with UNIX

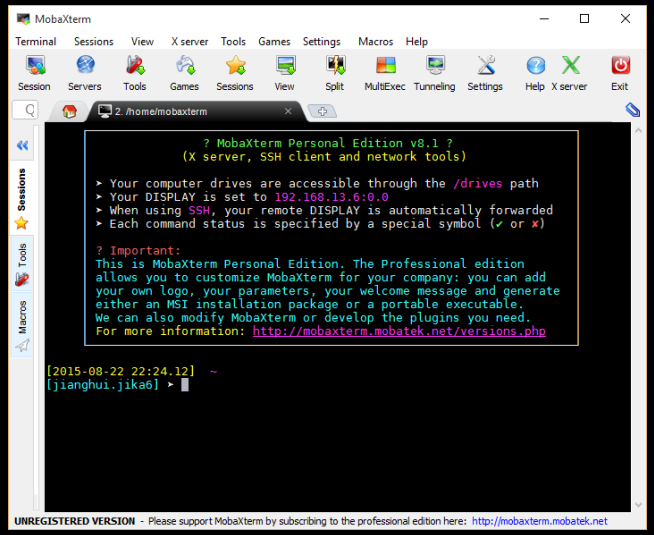
How do we start using UNIX?

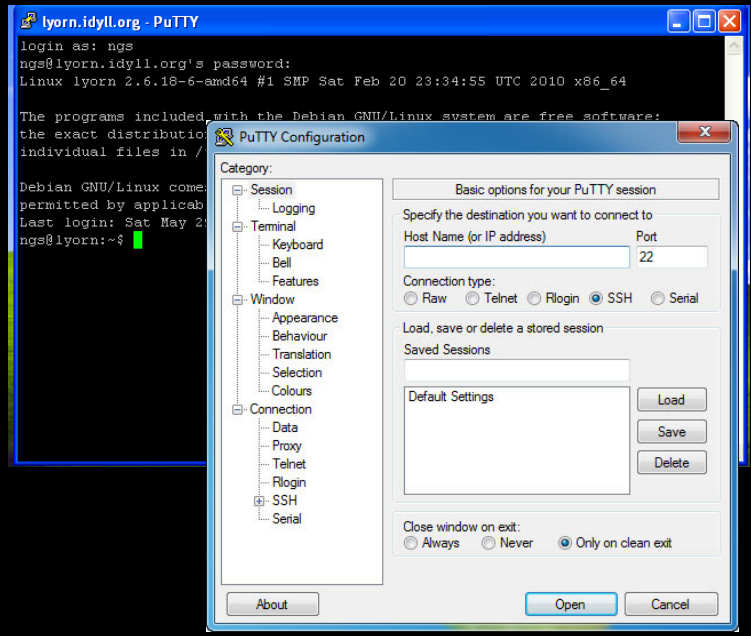
Getting Started

Mac OS X:
Terminal



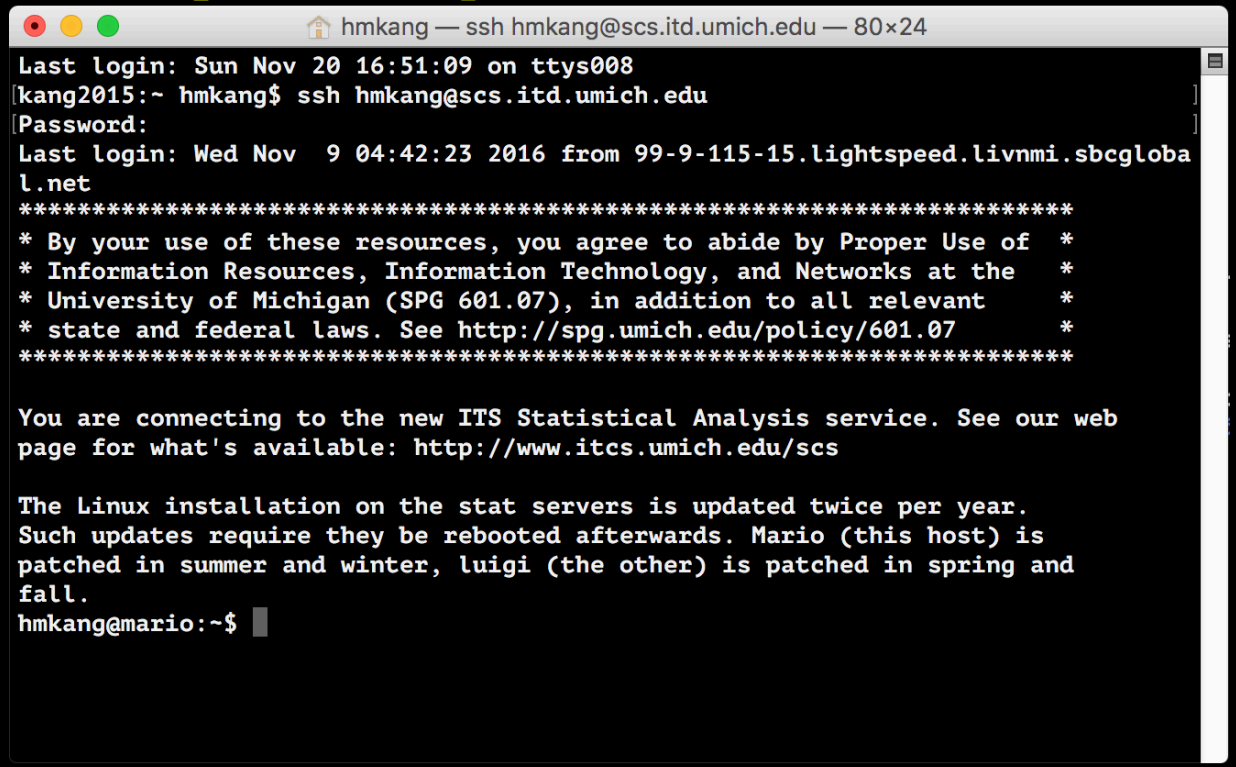
Windows:
MobaXTerm
or
PuTTY





Connecting to the Dedicated Server

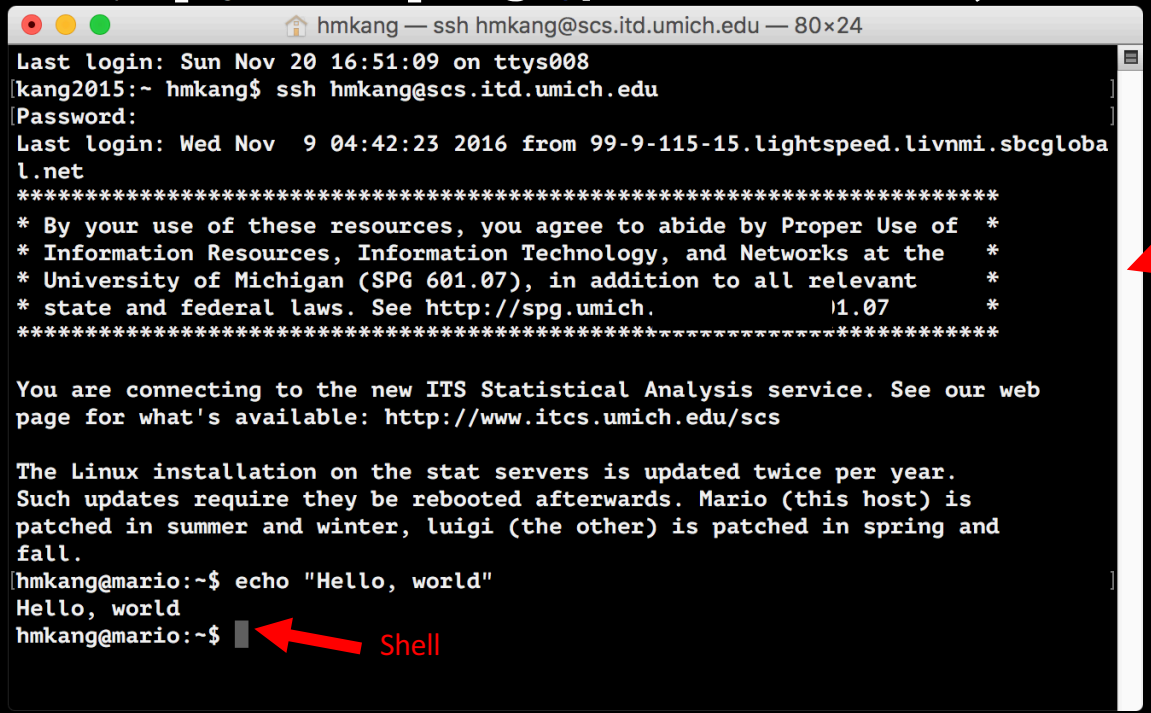
ssh [your uniqname]@scs.itd.umich.edu



type ‘bash [Enter]’ after logging in

Terminal vs Shell

- **Shell:** A command-line interface that allows a user to interact with the operating system.
- **Terminal** [emulator]: A graphical interface (window) to the shell.



What is a Process?

- **Process is a running instance of a program.**
- **Job: A group of processes (from a shell perspective).**
- **Some commands related to process/job:**
  - **`ps`**: report a snapshot of the current processes.
  - **`top`**: provides a real-time view of the running system.
  - **`Ctrl-c`**: stop a job.
  - **`Ctrl-z`**: suspend a job.
  - **`bg`**: resume a suspended job in the background.
  - **`fg`**: resume a suspended job in the foreground.
  - **`kill`**: terminate a process
  - **`[command]&`**: start a job in the background

[DIY] Try process-related commands

- Can you explain what each step does? -

1. Type `ps [Enter]`
2. Type `top [Enter]`
3. Type `Ctrl-c`
4. Type `top [Enter]`
5. Type `>`
6. Type `Ctrl-z`
7. Type `ps [Enter]`
8. Type `bg [Enter]`
9. Type `ps -ef [Enter]`
10. Type `fg [Enter]`
11. Type `Ctrl-z`
12. Type `bg [Enter]`
13. Type `ps [Enter]`
    - Find the `PID` of the 'top' process running background
14. Type `kill PID [Enter]`
15. Type `fg [Enter]`

File Systems in UNIX

- Information in the file system is stored in files, which are stored in directories (folders). Directories can also store other directories, which forms a directory tree.
- '/' is used to represent the root directory of the whole file system, and is also used to separate directory names.
- An absolute path specifies a location from the root of the file system (always starts with '/')
- A relative path specifies a location starting from the current location (never starts with '/')
- '..' means the parent directory (in a relative path)
- '.' means the current directory (in a relative path)
- '~' means the home directory (in a relative path)

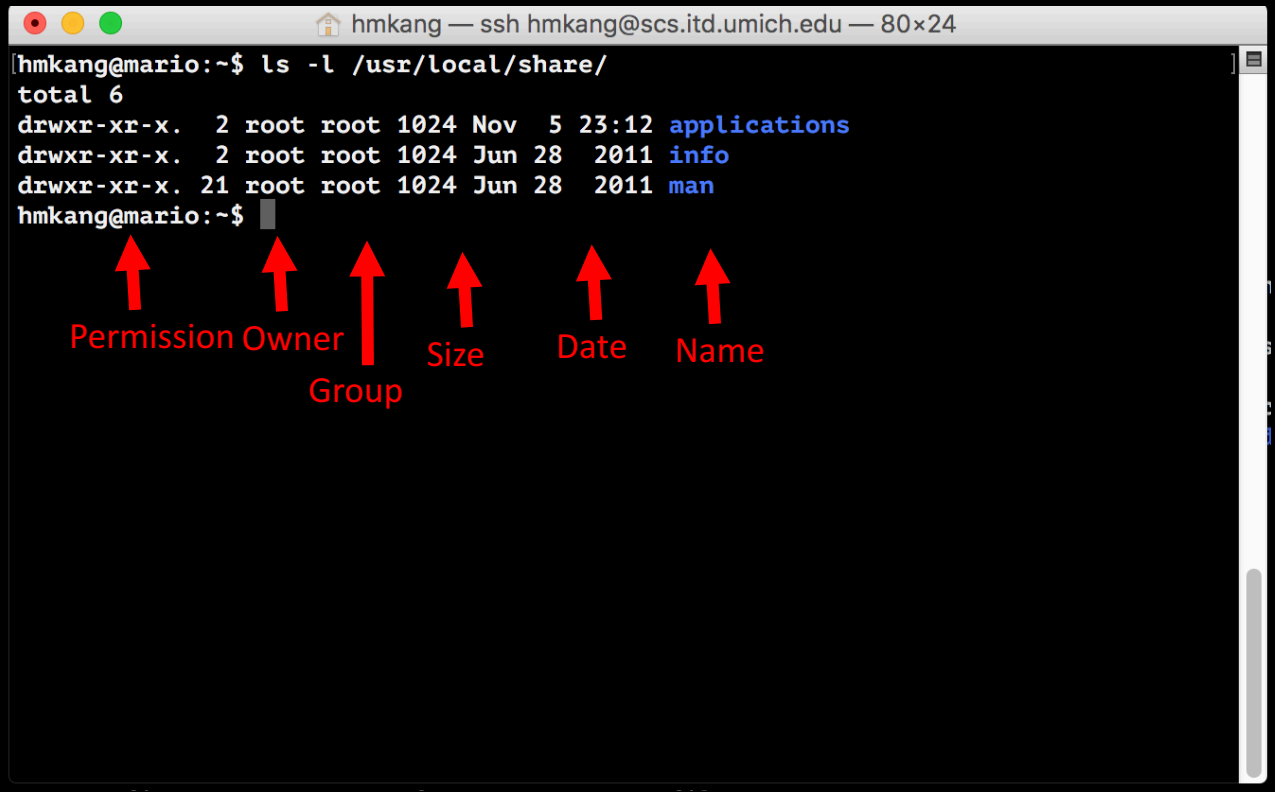
Commands Related to File Systems

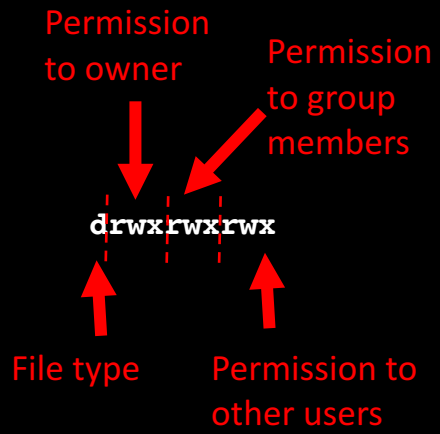
- `pwd`: print the user's current working directory.
- `cd`: change the current working directory.
- `ls`: print a listing of a specific file or directory.
- `cp`: copy files (or directories).
- `mv`: move files (or directories).
- `rm`: remove files (or directories).
- `mkdir`: create an empty directory.
- `touch`: create an empty file.
- '`*`': a wildcard matching zero or more characters in file (or directory) names.
- '`?`': a wildcard matching one character in file (or directory) names.

[DIY] Try directory-related commands

1. Type `pwd [Enter]`
2. Type `cd .. [Enter]`
3. Type `pwd [Enter]`
4. Type `cd ~ [Enter]`
5. Type `ls [Enter]`
6. Type `ls -l [Enter]`
7. Type `mkdir bag [Enter]`
8. Type `touch this [Enter]`
9. Type `cp this bag [Enter]`
10. Type `ls -l [Enter]`
11. Type `ls -l bag`
12. Type `touch that [Enter]`
13. Type `mv that bag [Enter]`
14. Type `ls -l [Enter]`
15. Type `ls -l bag`
16. Type `rm bag [Enter]`
17. Type `rm -rf bag [Enter]`
18. Type `ls -l [Enter]`

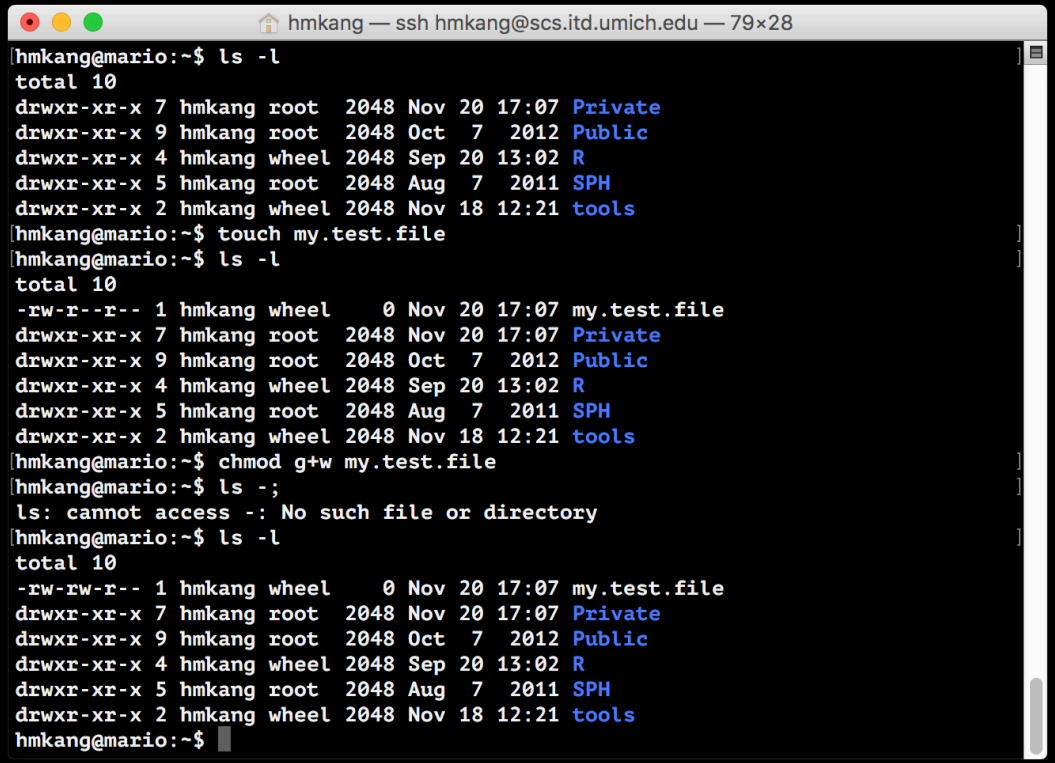
File Permissions





File Permission Related Commands

- `chmod`: change file permission.
- `chown`: change file owner and group.



Working with UNIX

We want more than basics!

Inspecting text files

- **`less` – visualize a text file**
  - Use arrow keys for basic navigation
  - Page down/up with **`[ space ]`** / **`[ b ]`** keys
  - Search by typing **`[ / ]`** key
  - Quit by typing **`[ q ]`** key

- **Also see**
  - **`head, tail, cat, more`**

[DIY] Try inspecting text files

1. Type `less /usr/share/dict/words [Enter]`
   (Use [tab] for auto-complete)
2. Type `[arrow keys]`
3. Type `[space] / [b]`
4. Type `[/] Linux [Enter]`
5. Type `[<]`
6. Type `[q]`
7. Type `head /usr/share/dict/words [Enter]`
8. Type `tail -n 20 /usr/share/dict/words [Enter]`
9. Type `cat /usr/share/dict/words [Enter]`

Creating text files

- **Creating files can be done in a few ways**
  - With `touch` command to create an empty file
  - With a text editor (`nano, emacs, vi`)
  - From the command line with redirection (`>`)

- **`nano` is a simple text editor recommended for first-time users**
  - Other text editors have more powerful features but also have steep learning curves.

[DIY] Creating and editing files with `nano`

- In the terminal, type

  ```
  $ nano somefilename.txt
  ```

- Write an acrostic poem on 'nano', and save the file, and exit

- View the contents of the file (how?)

- For long term use, pick one of the powerful editors.
  - `vi (vim)`
  - `emacs`
  - sublime text

[DIY] Finding the right hammer (`man` and `apropos`)

- You can access the manual (i.e. user documentation) on a command with man, e.g.

```
$ man pwd
```

- The man page is only helpful if you know the name of the command you're looking for. apropos will search the man pages for keywords

```
$ apropos "working directory"
```

- If you need to download a file, what command should you use?

Combining utilities with Redirection (`>`, `<`) and Pipes (`|`)

- The power of the shell lies in the ability to combine simple utilities (*i.e.* commands) into more complex algorithms very quickly.

- A key element of this is the ability to send the output from one command into a file or to pass it directly to another program.

- This is the job of `>`, `<` and `|`

Standard Input / Output / Error / Arguments

- **Two very important concepts that unpin UNIX workflows:**
  - Standard Output (**`stdout`**) - default destination of a program's output. It is generally the terminal screen.
  - Standard Input (**`stdin`**) - default source of a program's input. It is generally the keyboard input.

- **Two next important concepts**
  - Standard Error (**`stderr`**) – default destination of a program's error message. It is generally the terminal screen.
  - Arguments – Extra strings appended next to a program's name in the command line.

[DIY] Simple use of `stdin/stdout` with `tee`

1. Type `tee [Enter]` in the shell.
2. Type any text in the keyboard and press `[Enter]`
3. Try step 2 multiple times
4. If you're bored, press `Ctrl-c` to quit.

`tee` carries out a very simple task

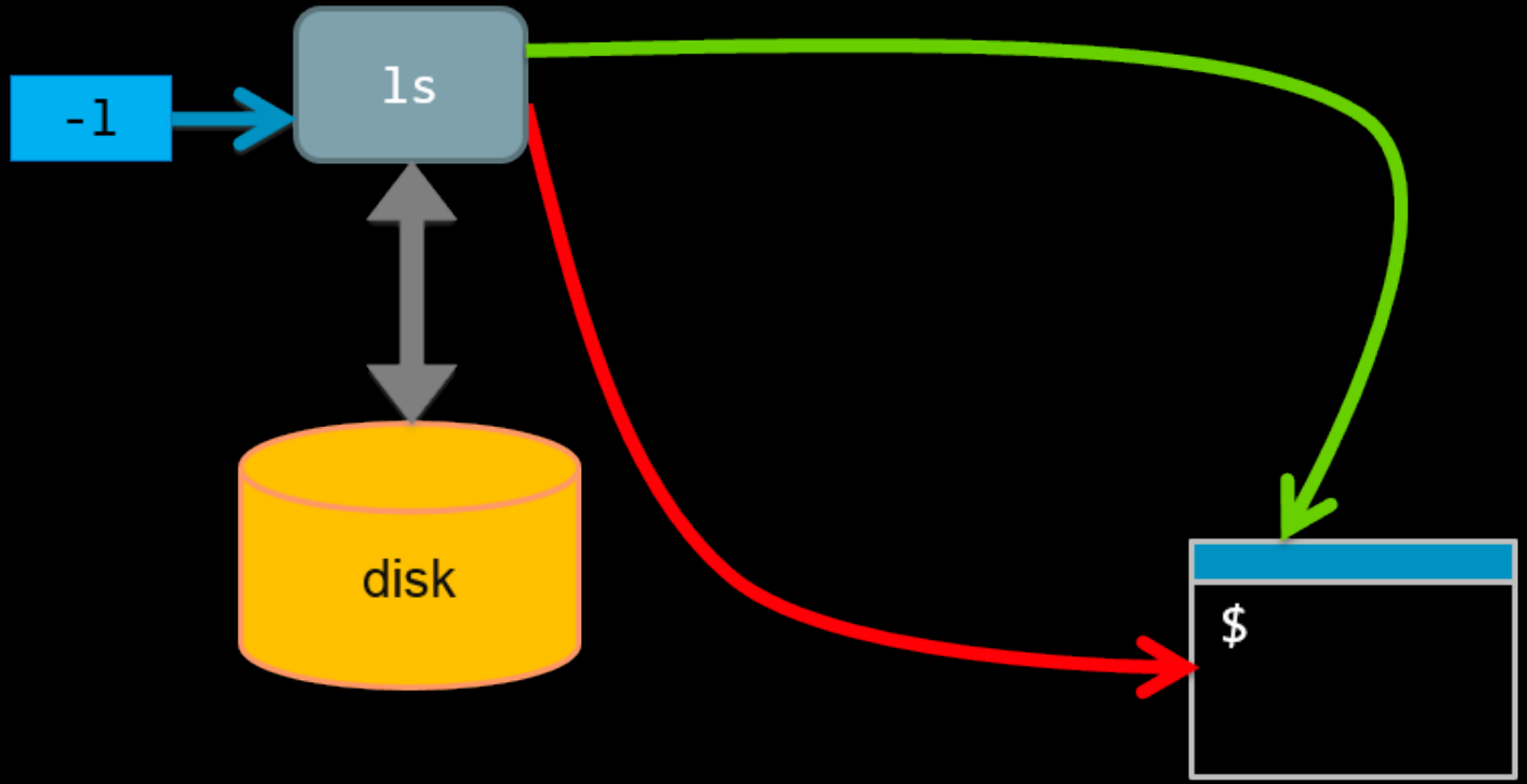
- It takes input from `stdin`,
- and copies it to `stdout`

Understanding the basic interface

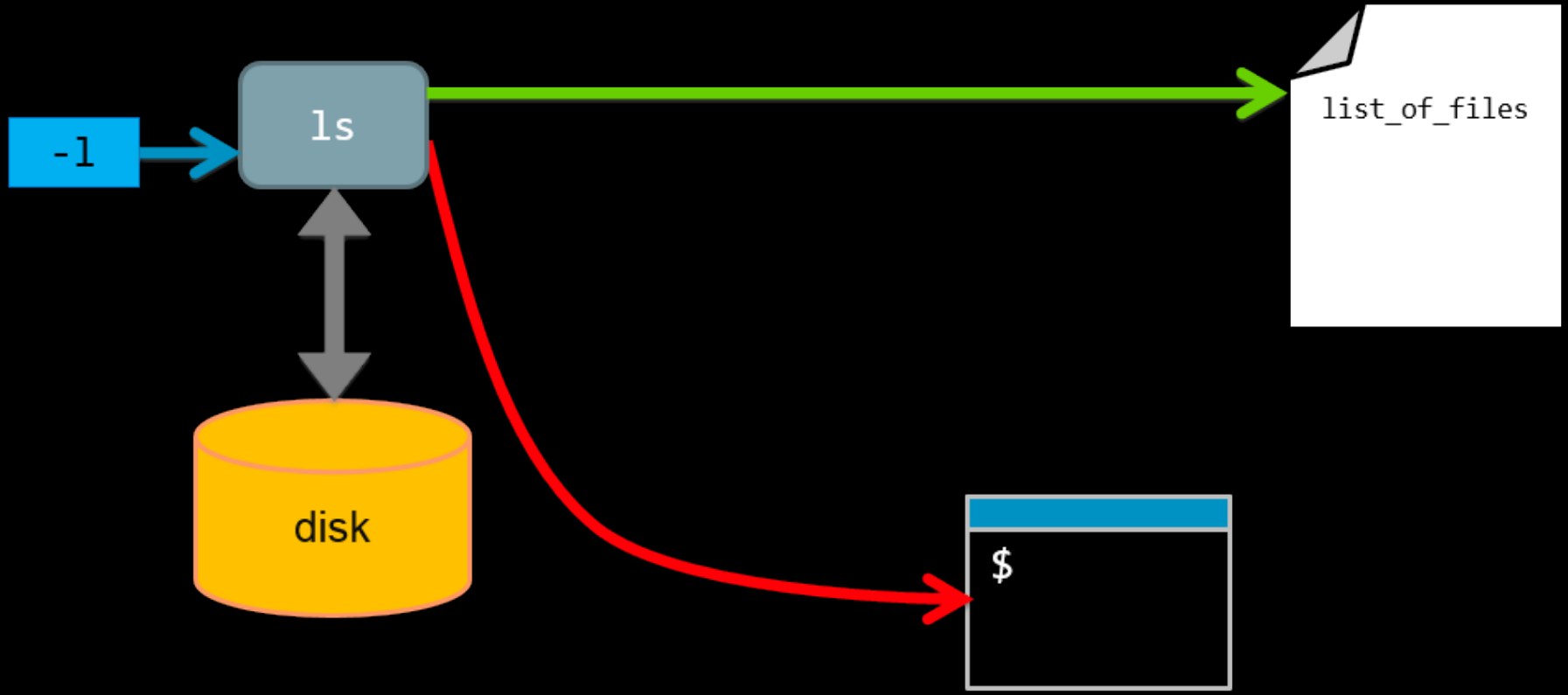
```
$ ls /usr/bin
```


```
$ ls /user/bin
```

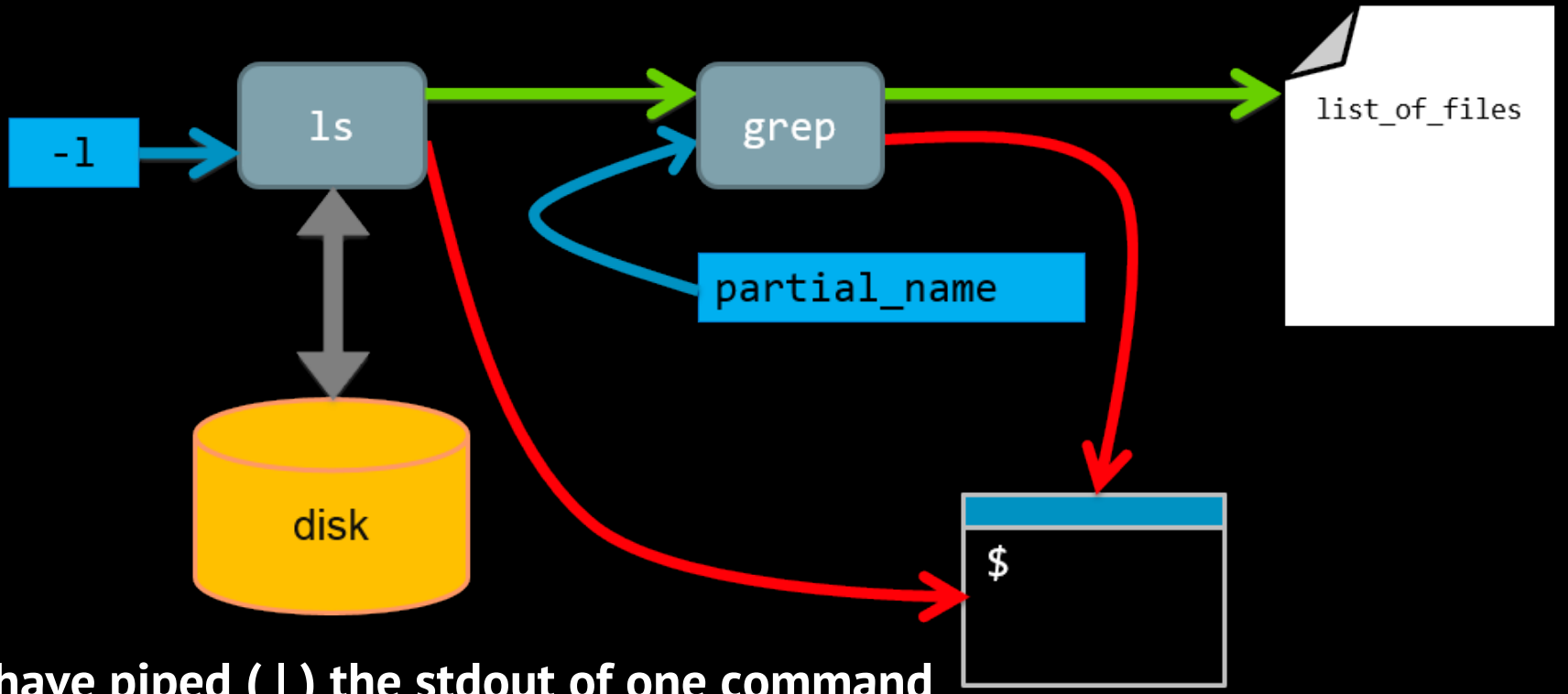
ls -l



ls -l > list_of_files

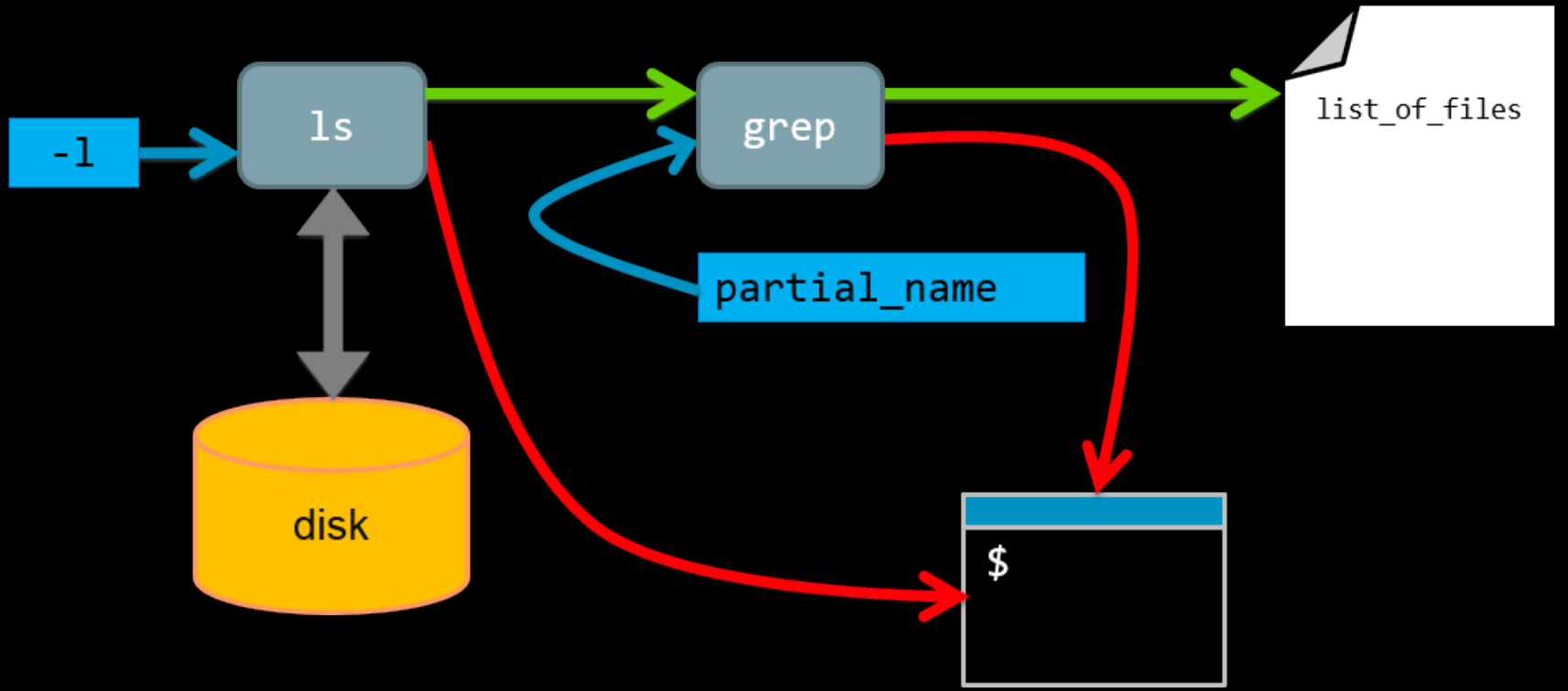


```
ls -l | grep partial_name > list_of_files
```



We have piped (|) the stdout of one command into the stdin of another command!

```
ls –l /usr/bin | grep tree > list_of_files
```



`grep` is a 'power-command'

- `grep` - prints lines containing a string pattern. Also searches for strings in text files

- For example,

```
$ grep apple /etc/dictionaries-common/words
```

- `grep` accepts regular expressions as input and has lots of useful options - see the *man page* for details.

`grep` examples using regular expressions

- Suppose that I want to search for all words starting with ‘tom’. You can eye-ball your file or …

```
$ grep tom /usr/share/dict/words
```

- Does this work as intended? Try to modify this way

```
$ grep “^tom” /usr/share/dict/words
```

- What if I want to search for words ending with ‘tom’?

```
$ grep “tom$” /usr/share/dict/words
```

`grep` with more regular expressions

- Using `-E` option, more sophisticated regular expressions can be used.

- Search for all words consisting only of ‘a, c, g, t’

```
$ grep -E '^[acgt]*$' /usr/share/dict/words
```

- Search for all words starting with ’y’ and also end with ‘y’

```
$ grep -E '^y.*y$' /usr/share/dict/words
```

Pipes and redirects avoids unnecessary I/O

- Disc i/o is often a bottleneck in data processing!

- Pipes prevent unnecessary disc i/o operations by connecting the stdout of one process to the `stdin` of another (these are frequently called "streams")

```
$ program1 input.txt 2> program1.stderr | \
  program2 2> program2.stderr > results.txt
```

- Pipes and redirects allow us to build solutions from modular parts that work with `stdin` and `stdout` streams.

Some more useful commands

```
$ wc -l /usr/share/dict/words

$ cut -c 1-3 /usr/share/dict/words | head

$ cut -c 1-3 /usr/share/dict/words | tr A-Z a-z | head

$ cut -c 1-3 /usr/share/dict/words | sort | head

$ cut -c 1-3 /usr/share/dict/words | sort | uniq -c | head

$ cut -c 1-3 /usr/share/dict/words | tr A-Z a-z| sort | uniq | wc -l
```

UNIX Philosophy

"Write programs that do one thing and do it well. Write programs to work together and that encourage open standards.
Write programs to handle text streams, because that is a universal interface."

— Doug McIlory

Pipes provides speed, flexibility, and sometimes simplicity

- In 1986 "*Communications of the ACM magazine*" asked famous computer scientist Donald Knuth to write a simple program to count and print the *k* most common words in a file alongside their counts, in descending order.

- Knuth wrote a literate programming solution that was 7 pages long, and also highly customized to this problem (e.g. Knuth implemented a custom data structure for counting English words).

- Doug McIlroy replied with one line:

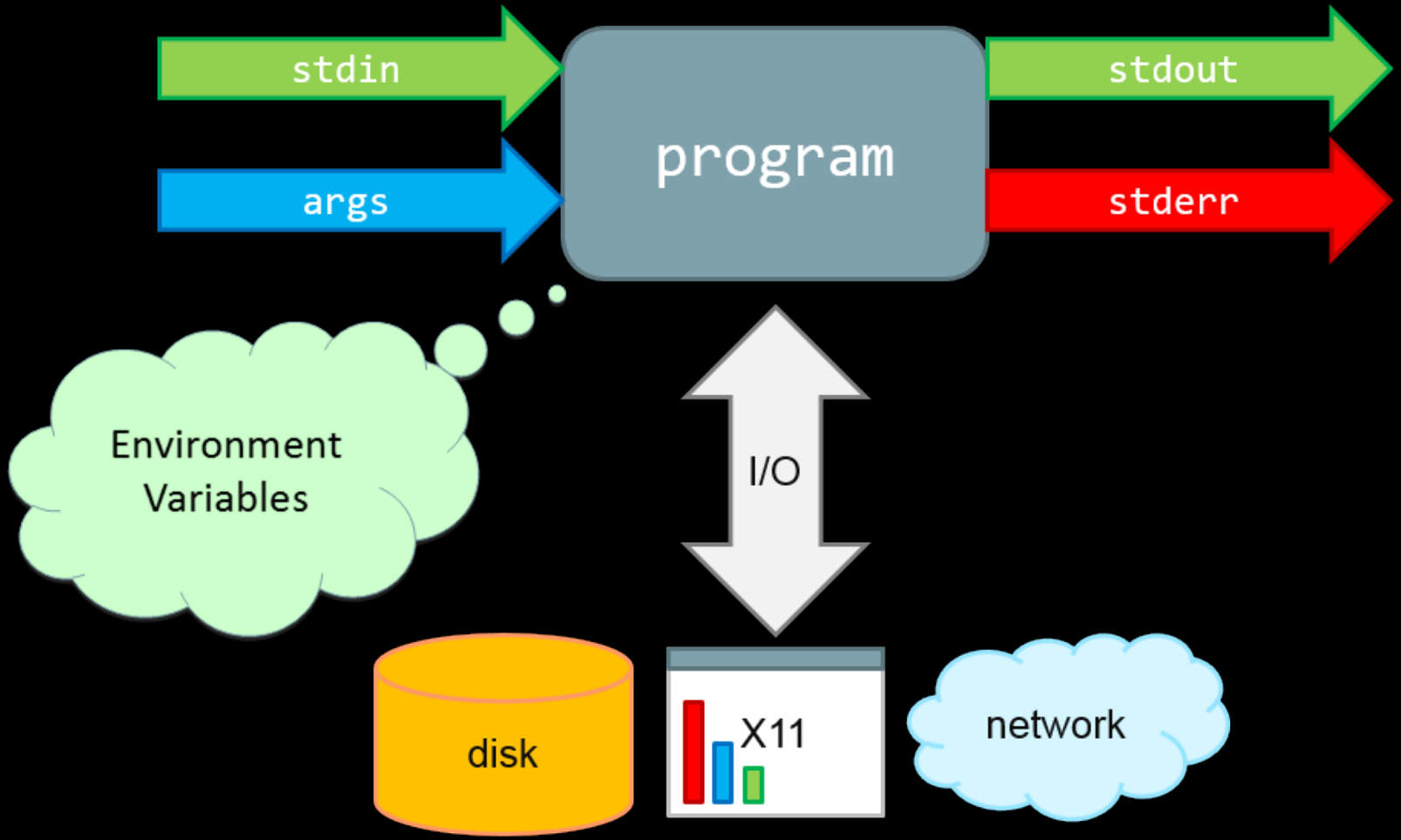
```
$ cat input.txt | tr A-Z a-z |  sort |  uniq -c |  sort -rn |  sed 10q
```

Key Point

- You can chain any number of programs together to achieve your goal

- This allows you to build up fairly complex workflows within one command line

Environment Variables



`$PATH` – a special environment variable

- What is the output of this command?

```
$ echo $PATH
```

- Note the structure: <path1>:<path2>:<path3>
- `PATH` is an environmental variable which Bash uses to search for commands typed on the command line without a full path.
- *Exercise*: Use the command env to discover more.

Summary

- Built-in unix shell commands allow for easy data manipulation (e.g. sort, grep, etc.)
- Commands can be easily combined to generate flexible solutions to data manipulation tasks.
- The unix shell allows users to automate repetitive tasks through the use of shell scripts that promote reproducibility and easy troubleshooting
- Introduced key unix commands that you will use during ~95% of your future unix work...

Useful Resources

- **UNIX reference (thanks to Barry Grant and Hui Jiang)**
  http://bioboot.github.io/web-2015/class-material/unix-reference.html

- **Guide to organizing computational biology projects**
  http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1000424

- **A tutorial to awk programming**
  http://www.grymoire.com/Unix/Awk.html

- **UC Berkeley's UNIX tutorial**
  http://people.ischool.berkeley.edu/~kevin/unix-tutorial/toc.html