



# Introduction to Linux Shell

**15-213/18-213/15-513/14-513/18-613:**

**Introduction to Computer Systems**



# Linux Shell

- The shell is a program that takes commands from the keyboard and gives them to the operating system to perform
- Computers “think” in text commands
  - Write commands using a “Command Line Interface” (CLI), often called a “terminal”
  - Say the basics of being in a directory, include slide

## What is linux?

- On most Linux systems a program called bash acts as the shell
  - Other shell programs which include: sh, ksh, tcsh and zsh.



# The Basics: Directories

Two commands commonly used to work with the current working directory:

- **pwd** – print working directory
  - This tells you what directory you are currently in
- **cd** - change directory
  - This lets you change into a different directory

## Important Directory Names:

- **~** – the home directory
- **~andrewid** – the home directory of user “andrewid”
- **.** – the current directory
- **..** – the parent directory (the directory right above the current one)
- **/** – the root directory (the main directory that has no parent)

---

# Using the Shark Machines



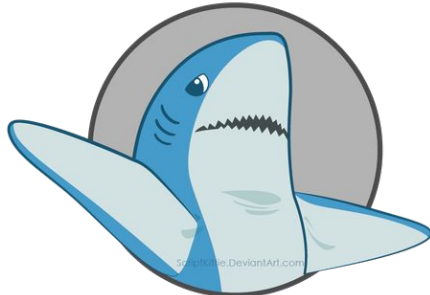
---

# Shark Machines

- In 213, we use shark machines which are used to access the Andrew File System [AFS] [linux.andrew.cmu.edu and unix.andrew.cmu.edu are other machines everyone has access to]
  - [15-213/18-213/15-513 Intro to Computer Systems: Lab Machines \(cmu.edu\)](#)

Why?

- They have correct versions of tools we will use and everything is set up to work in their environment



#TeamLeftShark

Without the shark machines, your code won't behave as expected!!!



# How can I access a shark machine? \*\*

ssh - a secure way to access any remote machine and execute commands.

1. `$ ssh ANDREW-ID@shark.ics.cs.cmu.edu`

(don't type in the "\$" this just means you're typing what follows into terminal)

1. `type your password when prompted`

\* If you see a warning about SSH host keys, click or enter "yes"

When you're done using a shark machine, be sure to logout!

`$ logout`

**\*\* Please use shark machines for ONLY 213, we need to save their computing power for this course. Otherwise, you should use cmu linux machines \*\***



# Manual pages (man pages)

- If you are ever unsure about a command one helpful resource is to utilize man pages
- `$ man <command>`
  - Gives information on what a command does and what options you can give it.
- You can search through a man page by typing: `/thing_i_want_to_find`
  - Advance from one match to the next by pressing `n`
- Most commands have a `--help` or `-h` option that will print out a help message

For more information about the man command, enter:

```
$ man man
```



# Transferring files between machines

scp: a secure way to copy files between 2 machines

```
$ scp user@alpha.com:/somedir/somefile.txt user@beta.com:/anotherdir
```

Remote to Local      `$ scp username@from_host:file.txt /local/directory/`

Local to Remote      `$ scp file.txt username@to_host:/remote/directory/`

Remote to Remote    `$ scp username@from_host:/remote/file.txt username@to_host:/remote/directory/`

## Flags

- r: recursive [useful for copying directories]
- v: verbose mode [useful for debugging]
- q: quiet [useful for when updated messages are not needed]

NOTE: If you are copying a file to a current directory, use `.` as the file path. If you are recursively copying a directory from your local machine, use `.` as the file path. See slide about **DIRECTORIES** for more.



---

# Managing your files



# Managing files: Moving, creating & deleting files

- `cp <source> <destination>` - copy files
- `mv <source> <destination>` - move and rename files
- `rm <filename>` - PERMANENTLY delete files
- `rmdir <filename>` - PERMANENTLY delete empty directory
- `mkdir <directory>` - make directories
- `touch <file>` - create an empty file
- **List the files in the current directory:**
  - `ls [path]` - listing files
  - `tree [path]` - recursively listing files



## WHAT NOT TO DO [BAD]

```
$ rm -rf /  
$ rm -rf *  
$ rm -rf .  
$ mv /home/user/* /dev/null
```

- [What Not to Do Part 1](#)
- [What Not to Do Part 2](#)



**THESE ARE**  
**BAD! NEVER**  
**TRY THIS!**

# Hidden & Temporary Files

```
E325: ATTENTION
Found a swap file by the name ".hat.txt.swp"
  owned by: alhoffma   dated: Sun Jun 14 09:12:24 2020
  file name: ~/alhoffma/private/hat.txt
  modified: YES
  user name: alhoffma   host name: unix6.andrew.cmu.edu
  process ID: 23658
while opening file "hat.txt"
  dated: Sun Jun 14 09:12:09 2020

(1) Another program may be editing the same file.  If this is the case,
    be careful not to end up with two different instances of the same
    file when making changes.  Quit, or continue with caution.
(2) An edit session for this file crashed.
    If this is the case, use ":recover" or "vim -r hat.txt"
    to recover the changes (see ":help recovery").
    If you did this already, delete the swap file ".hat.txt.swp"
    to avoid this message.

Swap file ".hat.txt.swp" already exists!
[O]pen Read-Only, [E]dit anyway, [R]ecover, [D]elete it, [Q]uit, [A]bort:
```

- Hidden files begin with a . and are hidden unless you specify a command for -a (all)

## Swap Files **\*\*applies to vim\*\***

`malloc.c`                      `.malloc.c.swp`

- A copy of an old version of a file that was not properly saved
- Solution:
  - Delete swap file from command line

### Open Read Only

Useful for when you only want to view contents

### Edit anyway

Be careful! If the file is being edited in another vim session, you will have 2 versions

### Recover

Useful for when you know the swap file contains the changes you want to recover

### Delete it

Useful for when you no longer need the file

### Quit

Useful to not edit the current file but want to keep other vim sessions open

### Abort

Useful to close any open vim sessions



# Tar

A way to archive files in 1 bundle (and compress them)

## Flags

-c: create a tarball

```
tar -cf name-of-archive.tar /path/to/dir/  compress directory
```

-x: open a tarball

```
tar -xf name-of-archive.tar /path/to/filename  compress file
```

-z: zipped using gzip

```
tar -czf name-of-archive.tar dir1 dir2 dir3  compress multiple directories
```

-v: verbose mode [displays progress]

```
tar -xvf name-of-archive.tar  open a tar file in current directory
```

-f: specify file name

**This might be helpful for bootcamp labs!**



# Text Processing

- \$ **sort** sort lines of text file
  - d: dictionary order
  - f: ignore case
  - n: numeric sort
  - r: reverse
- \$ **cat** concatenates inputs and prints on the screen
- \$ **uniq** reports or omits repeated lines
- \$ **head/tail** prints first (or last) lines of a file
  - bx: print out first (or last) x bytes
  - nx: print out first (or last) x lines

---

# Other helpful aspects of a shell



# File Redirection

## Syntax

```
command < file.txt
command > file.txt
command >> file.txt
command 2> file.txt
command 2>> file.txt
```

## Meaning

```
Read the stdin of “command” from “file.txt”
Send the stdout of “command” to “file.txt”, overwriting its contents
Append the stdout of “command” to the end of “file.txt”
Send the stderr of “command” to “file.txt”, overwriting its contents
Append the stderr of “command” to the end of “file.txt”
```

## Example\*:

```
# 'hello.txt' doesn't exist, so it will be created
$ echo "Hello" > hello.txt
$ cat hello.txt
Hello
```

\*more on echo at the end :)





# Grep (Global Regular Expression Print)

grep searches for patterns in a file [if no file is provided, all files are recursively searched]

```
$ grep [OPTION...] PATTERNS [FILE...]
```

## Standard Flags

- c: prints count of matching lines
- h: display matches without filenames
- i: ignores case for matching
- l: displays list of only filenames
- n: display matches and line numbers
- e **exp**: specifies expression with this option
- f **file**: takes pattern from file
- o: print only matching parts of lines
- r: read all files under each directory, recursively

## Examples

```
$ grep "test" *  
$ grep -r "test"  
$ grep -rc "test"
```



# Pipes (|)

- Pipes are a way to chain together the output from one command with the input to another.
  - To create a pipe, we use the Unix pipe character: |
- Lets use `grep` to find words in the computer's dictionary that contain the string "compute"
- ```
$ grep compute /usr/share/dict/words
```

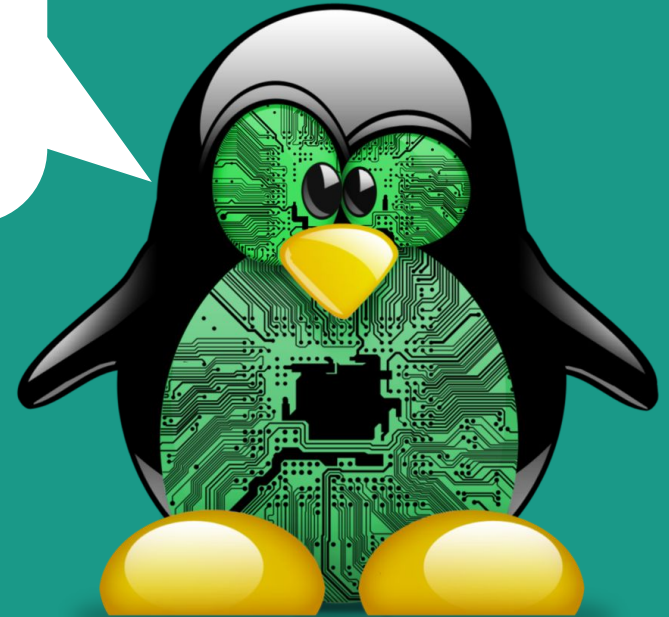
```
>> compute
    computer
    ...
    Uncomputed
```

```
# Pipe output of grep (on stdout) to the input of wc (on stdin)
$ grep "compute" /usr/share/dict/words | wc -l
```

```
>> 34 # Thus, 34 words have the word 'compute' in them
```
- Using pipes effectively can reduce some incredibly hard problems down to one line of code

---

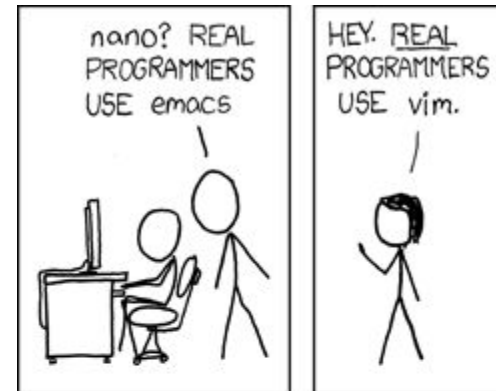
# Writing Code on Shark Machines



---

## Option 1: VIM

- Can be run on pretty much any terminal.
- To make vim *\*spicy\** or to add extra functionality you can modify your `~/.vimrc!`
- Use vim by SSH-ing into a shark machine
- According to legend, if you learn all the keyboard shortcuts, the rate at which your fingers travel approach lightspeed, to the point of being a potential hazard to those in your general vicinity.





## Option 2: VS Code + SFTP

- Text editor with lots of traditional functionality.
  - Tabs, easy window split, built-in terminal, tree view, etc.
- Cool plugins to make code prettier + life easier.
- People won't make fun of you for using the mouse.
  - (Except for VIM purists)
  - (Don't let the haters get to you)

### HOTTEST EDITORS

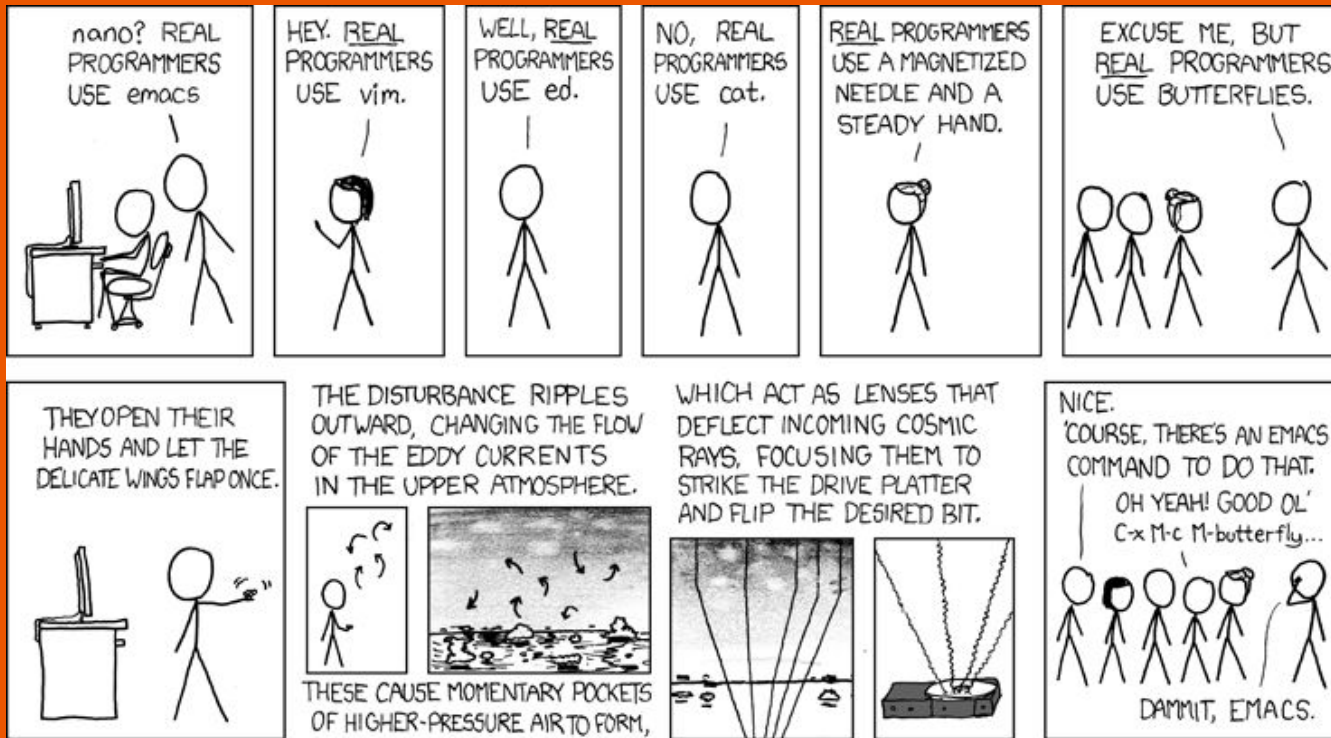
1995 — [EMACS-VIM]  
2000 — [EDITOR WAR]  
2005 — VIM  
2010 — NOTEPAD++  
2015 — SUBLIME TEXT  
2019 — VS Code  
2025 — FB Messenger





## Other Notable Editors

- Emacs
  - Command line editor like Vim
  - Lots of powerful features, but can be very complicated at times
- Sublime Text
  - Editing application with an SFTP Plugin, like VS Code
  - Simpler and more lightweight than VS code



—

**But seriously, use vim.**  
or use something else



# Vim!

- Vim is a terminal-based text editor that is highly customizable
- `$ vim ~/.vimrc`
- Three big modes:
  - Normal mode: press the “esc” key.
  - -- **INSERT** -- mode: press the “i” key in normal mode (make sure you see the -- **INSERT** --)
    - Then type that into the text buffer
  - -- **VISUAL** -- mode: press the “v” key in normal mode.
    - Use arrow keys to highlight a selection
    - “Copy and paste”:
      - Highlight text, press “y” to yank (copy) and “p” to paste
      - Press “d” twice will delete the selection, which also makes it available to paste with “p”





## Closing Vim :(

- When done, press “esc” and then type in “:w” to save
- Type in “:q” to quit VIM
- This can be combined into “:wq” to save and quit in one command

Try \$ [vimtutor](#) for more tips and tricks for VIM that we might not have covered!

---

**Lab Time!**

**<https://tinyurl.com/15213bootcamp1>**

# What is git?

- git ≠ GitHub
- Version control system
  - Better than:
    - copy pasting code
    - emailing the code to yourself
    - taking a picture of your code and texting it to yourself
    - zipping the code and messaging it to yourself on facebook
- **using git this semester will (with high probability) be mandatory!!!**  
~\*style\*~ point deductions if you don't use it





## Configuring git

```
$ git config --global user.name "<Your Name>"
```

```
$ git config --global user.email "<Your Email>"
```

```
$ git config --global push.default simple
```

(Make sure the email is your Andrew ID, and make sure to add that email to your GitHub account!)

To see the current set of configs: `$ git config --list`



## Creating a Repository

- `$ git init`            turn the current folder into a new repository.

OR

- `$ git clone`            initialize a repository locally from a remote server

For example, in `~/private/15213`:

```
$ mkdir datalab
```

```
$ cd datalab
```

```
$ git init
```



## Core Gameplay Loop

1. `$ git add` Stages files to be committed. Flags: `--a` (all files), `-u` (only previously added files)  
Can also add specific files by listing them after 'add'.
1. `$ git commit -m "<MESSAGE>"` Commit the changes in the staged files. Write descriptive, meaningful messages for future you!

If repository is connected to remote server (like GitHub):

1. `$ git push` Push changes to a remote server.

If working with others on remote server:

0. `$ git pull` Pull changes from a server



## Other Important Commands

- `$ git status` shows key information such as current branch, “add”ed files, “commit”ed files not yet pushed.
- `$ git log` show commit history. Can use `--decorate --graph --all` to make it pretty.
- `$ git diff` shows the changes you’ve made
- `$ git rm` stages files to be removed.
- `$ git reset HEAD <FILE_NAME>` unstages “FILE\_NAME” from the commit

Documentation for all commands: [Git - Documentation \(git-scm.com\)](https://git-scm.com/docs)



# Chronomancy (The Greatest Magic of All!)

A Time Lord's toolkit:

- `$ git revert <COMMIT_HASH>`
  - Creates a new commit where everything is the same as the commit with COMMIT\_HASH
- `$ git checkout <FILE_NAME>`
  - Used to reset any changes made to a file to previous commit.
- `$ git reset --hard <COMMIT_HASH>`
  - Sets you back to COMMIT\_HASH.
  - Commits between COMMIT\_HASH and present time will disappear.
- Many more ways to do similar and different things, all in Git documentation.





# Time Magic is Dangerous!

- Uncommitted changes will be destroyed.
- Often cannot be undone
- `$ git reset --hard`
  - Going back multiple commits will rewrite history and make crew members time-sick.
  - Last resort that should only be used in private repos. Revert is safer.



# ~Helpful (& Miscellaneous) Tips & Tricks~



## What is AFS?

- Shark machines use AFS [Andrew File System] (a distributed file system) to store files
- AFS Machines have two default directories where you write code:
  - `~/private`: where you can store your classwork
  - `~/public`: where others can read files and copy them but not change, delete, or add files

WRITE YOUR CODE IN YOUR `~/private` DIRECTORY!



# Permissions

- `fs la`: to understand permissions for a directory
- Permissions:
  - **r** read
  - **w** edit existing files
  - **l** list files and see basic information
  - **k** “lock” files so no one can edit them at the same time
  - **i** create new files
  - **a** admin
  - **d** delete files
- `fs sa <directory> <user> <permission>`: how to set permissions on a file or directory

```
$ fs sa foo acarnegie rldwik
```



# Recovering Lost Files

Oh no, I've deleted all of my files!

- Plan A: Git version control (revert old commit) or Github
- Plan B: CMU keeps a nightly snapshot of files in ~/OldFiles
- Plan C: Run the following command to create ~/OldFiles from backup:

```
$ cd ~  
$ fs mkmount OldFiles user.ANDREW_ID_HERE.backup
```



# Wildcards

- \*: Matches any characters
- ?: Matches any single character
- [ characters ] \*: Matches any character that is a member of the set *characters*.
- [ ! characters ]: Matches any character that is NOT a member of the set *characters*.
- More info: [Wildcards \(tldp.org\)](http://tldp.org)

```
rm g*
```

remove all files starting with “g”

```
ls b*.txt
```

list all files that begin with “b” and end with “.txt”

```
cat Data???
```

cat any file beginning with Data and has exactly 3 more characters

```
tree [abc]*
```

tree any files that begin with “a” “b” or “c”

```
[[ :upper: ]]*
```

references any file beginning with an upper case character

```
*[![:lower:]]
```

references any file that does not end with a lowercase letter

\*this also works with any POSIX character classes!



# Processes

- A process is an instance of a running program.
  - Not the same as “program” or “processor”
- A process is a currently executing command (or program), sometimes referred to as a job.
- At any given time there may be a couple hundred or less processes running.
- If you’re running Linux or a Unix based machine you can run a number of different commands:
  - `ps aux` // this will display a list of processes
  - `top` // detailed information about all processes, threads, and more



# Foreground vs Background Jobs

A job is a process that is currently running or has been stopped or terminated.

- Unique job ID (JID) to each job, and can be run in the **foreground** or the **background**
  - Foreground job: a job that occupies the terminal until it is completed
  - Background job: a job that executes in the background and does not occupy the terminal
    - A background job can be run by writing a & at the end of the line
- The shell can only handle 1 foreground job and many background jobs at the same time





# Commands related to jobs

- `jobs`: lists the state of all jobs
- `fg %n`: brings current or specified job in foreground; n is JID
- `bg %n`: places current or specified job in background; n is JID
- `CTRL + z`: stops foreground job and places it in the background as a stopped job [this job can be restarted later]
- `CTRL + c`: sends SIGINT to a foreground job and usually causes it to exit [it can never be restarted]

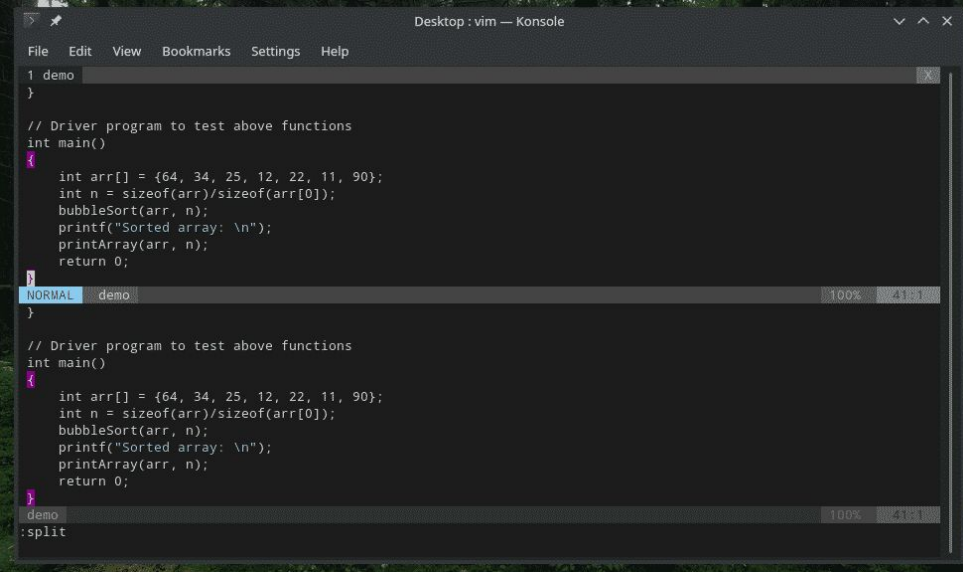


# Echo & sed commands

- **echo** is used to display line of text/string that are passed as an argument
- `$ echo ls -l | sh`
  - #Passes the output of "echo ls -l" to the shell, with the same result as a simple `ls -l`
- **sed** can do insertion, deletion, search and replace (substitution)
- `$ sed 's/old_word/new_word/' file.txt`
  - s → substitution
  - Substitutes 'old\_word' with 'new\_word' in file.txt

# Vim Tricks: Split

- `:sp <path_to_file>/filename.extension` to split a vim terminal with 2 files
- CTRL + W then j goes to lower window
- CTRL + W then k goes to higher window
- You can use standard vim controls to navigate within each window



The screenshot shows a terminal window titled "Desktop : vim — Konsole". The terminal is in split mode, displaying two identical copies of a C program. The top window shows the first few lines of the program, and the bottom window shows the same code. The status bar at the bottom of the terminal indicates the current mode is "NORMAL" and the file is "demo". The command `:split` is visible at the bottom of the terminal.

```
Desktop : vim — Konsole
File Edit View Bookmarks Settings Help
1 demo
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}

NORMAL demo 100% 41:1
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}

demo 100% 41:1
:split
```

---

# Using VS Code & SFTP

## Option 2: VS Code + SFTP

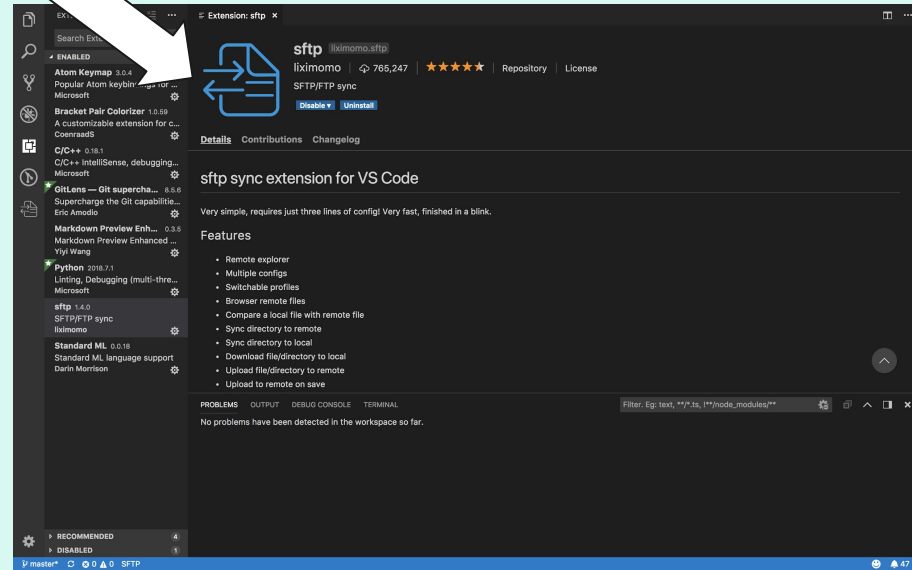
- Secure File Transfer Protocol
- Can be used to read from and write to files in your AFS directory by synchronizing changes to files saved directly to your machine
- **Note on academic integrity:** since the files exist on both your machine and AFS, you are responsible to make sure that no one gets access to either location!



## Option 2: VS Code + SFTP

### On your local machine:

- Download VS Code here:  
<https://code.visualstudio.com/download>
- You can check out some of the other extensions (linting for C for style???) on your own time, but download Natizyskunk's SFTP plugin because that's how we're gonna be working with the Shark machines.



## Option 2: VS Code + SFTP

### On your local machine:

- Create a local 213 folder.
- Inside there, create a folder called “linux-bootcamp.” Open it in VSCode.
- Ctrl + Shift + P (Windows) or Cmd + Shift + P (Mac) to open up Command Palette:
- Type in “SFTP: Config”
  - This should open “sftp.json”
- Type in the info in the top-right.
  - **Notes:** “uploadOnSave” will automatically save any local changes whenever you save the file. “downloadOnOpen” will automatically update your local version with the AFS version when you open the file.
- Visit <https://github.com/Natizyskunk/vscode-sftp/wiki/Configuration> for extra config options

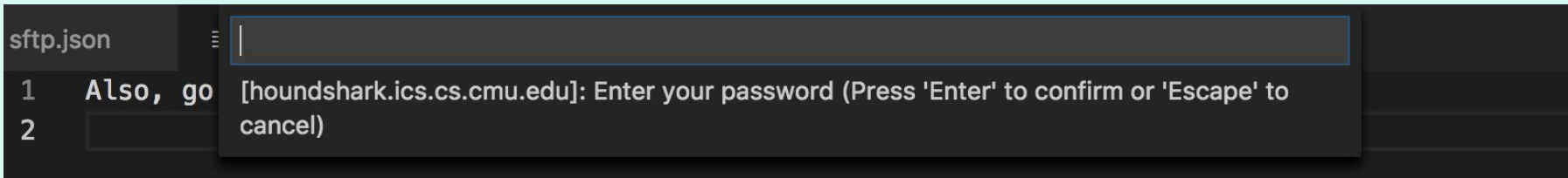
```
{
  "name": "<this-can-be-anything>",
  "protocol": "sftp",
  "host": "<your-favorite-shark-machine>.ics.cs.cmu.edu",
  "port": 22,
  "username": "<andrew-id>",
  "remotePath": "/afs/andrew.cmu.edu/usr3/eyluo/private/15213/linux-bootcamp",
  "uploadOnSave": true,
  "downloadOnOpen": true,
  "ignore": [
    ".vscode",
    ".git",
    ".DS_Store",
    "admin"
  ]
}
```



## Option 2: VS Code + SFTP

### On your local machine:

- Create a file called “example.txt” and type whatever you want into it.
- When you save, this should prompt a popup to type in your SSH password.



The screenshot shows a VS Code editor window with a file named 'sftp.json'. The editor content includes the text 'Also, go' followed by a line number '2' and a text input field. A modal dialog box is overlaid on the editor, displaying the prompt: '[houndshark.ics.cs.cmu.edu]: Enter your password (Press 'Enter' to confirm or 'Escape' to cancel)'. The dialog box has a dark background and a light border.





## Option 2: VS Code + SFTP

### On your favorite Shark machine:

- SSH into your favorite Shark machine.
- `$ cd` into your example directory and list the files. You should see “example.txt” inside!



## Option 2: VS Code + SFTP

### Reminders:

- SFTP means you're downloading code from AFS onto your local machine, so take extra precaution to make sure that code is secure and no one steals it!
- Any time you run `$ make`, please do so on the Shark machines!!



# Sources?

<https://www.tldp.org/LDP/abs/html/textproc.html>

<http://linuxcommand.org/lc3 Its0010.php>

<https://www.cs.cmu.edu/~15131/>

<https://www.cs.cmu.edu/~213>

<https://haydenjames.io/linux-securely-copy-files-using-scp/>

---

**Feedback:**

**<https://forms.gle/ohKmjJFQKoEbqcA36>**