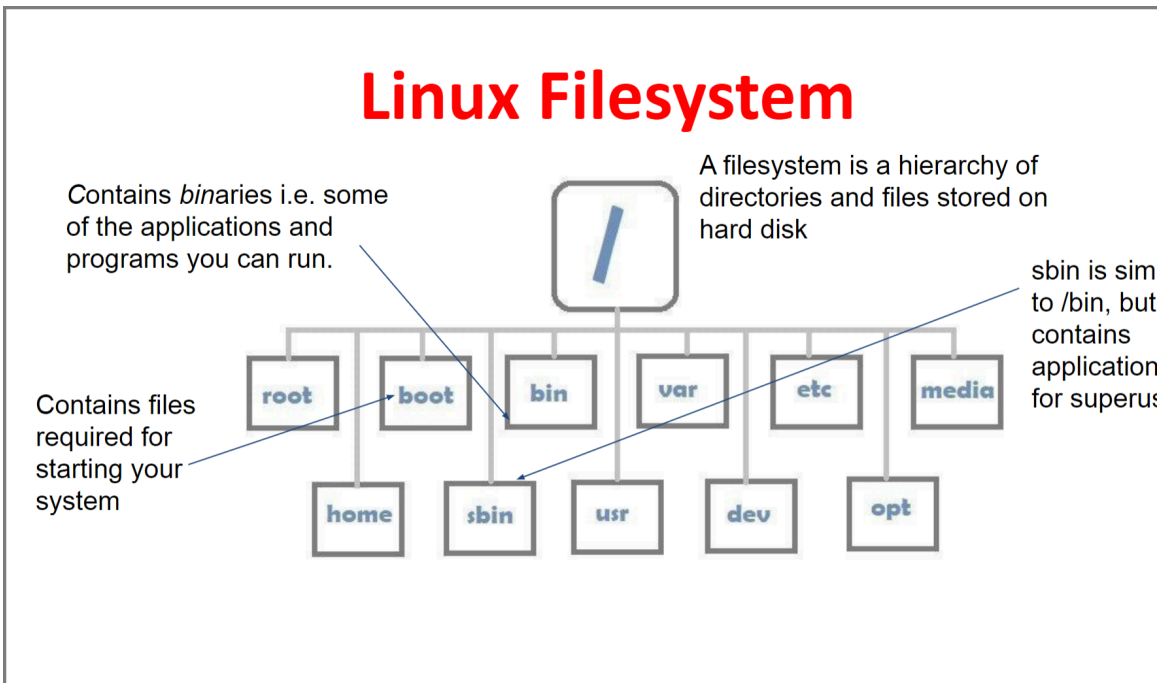
1. 

- "/" at the start of your path means "starting from the root directory"
- ("~") at the start of your path means "starting from my home directory"

ls:

- -l gives list format of files present
- -a displays all files, including hidden ones.
- -R recursive
- -S sorts by file size, -X by name.

- mkdir -p checks whether parent folders are there and creates them accordingly.

cat:

- -n gives the output with file numbers.
- "less" gives lesser content
- -s to omit repeated empty lines.

mv:

- mv {options} {source} {destination}
- source can be multiple files, single file or directory(ies)
- if source is a single file, and dest doesn't exist, renaming happens
- if source and dest exist, both being files, the source is deleted and the contents is put in dest
- if source is dir and destination dir doesn't exist, source gets renames.
- -i prompts before moving, -n prevents over writing, -f makes sure on prompt comes up while moving.

cp:

- copies files from source to dest

rm:

- -i prompt before removing
- -r does the removing recursively, can be used to delete a dir completely even if it has stuff innit.
- -d removes dir, same as rmdir.

- head displays the first few lines, tail displays the last few.
- ```
  head -n 3 students.txt
  ```

```
mkdir Vehicles/{Cars,Trains,Aeroplanes,Ships}
```

to create a directory structure.

REGEX:

Metacharacters: characters with special meaning

- "^" beginning of a line (Can also mean "not" if inside [])
- "$" end of line
- "." match any single character
- "\" escape a special character
- "|" or operation i.e. match a particular character set on either side

Quantifiers: specifying the number of occurrences of a character

- "*" Match the preceding item zero or more times
- "?" Match the preceding item zero or one time
- "+" Match the preceding item one or more times
- "{n}" Match the preceding item exactly n times
- "{n,} Match the preceding item at least n times
- "{,m}"Match the preceding item at most m times
- "{n,m}" Match the preceding item from n to m times

Groups and Ranges

- " ( )" group patterns together

"[ ]" match any character from a range of characters
```

[^.....] matches a character which is not defined in the square bracket

grep:

- grep -i "for a" bigfile (here -i ignores the case and newline, so if -i wasn't given then for\n a wouldn't be considered)
- in place of "for a" we could also have regex.
- grep -E "hi|hey" file (searches for hi or hey in the file, -E enables the escape seq)
- \s indicates a whitespace.
- -v negation, -r recursive search, -w whole word, -n show line number, -c count (count of lines, not number of occurrences -e uses patterns as patterns(pipe will be interpreted as a pipe)
- \b matches any character that is not a letter or number
- \d represents any digit.

find:

- locates file/dir with known pattern
- find {source} -name "{filename}" searches for files with given name
- find source -type f tells the terminal to search only for files. (-d for directories)
- can also find based on size, perms, and can also delete the stuff found.

wc:

wc: print word, character, line, byte count in a file

- can accept zero (reads from standard input) or more input
- Default output: lines, words, characters
- "-l" number of lines.
- "-w" number of words.
- "-m" number of characters.
- "-c" number of bytes

- can have multiple files as argument.

sort:

- -r does reverse, -n does numerically
- when dealing with csv files, -t for specifying the delimiter, -k for selecting the field you want.

zip/unzip:

- -r zips recursively, -e encrypts it.
- syntax: zip {name of zip} {source}
- unzip -l just lists the stuff inside the zip, doesn't actually unzip.
- unzip {the zip} <-d dir1> (puts the unzipped stuff in dir1)

tar:

- similar to zip but for unix
  - -c: Create a new archive.
  - -v: Verbose mode
  - -f: Specify the name of the archive file
  - -x: Extract files from an archive.
  - -z: Specifies that the archive is compressed with gzip
  - -t: Lists the contents of the archive
-
  - To create a TAR archive: tar -cvf archive.tar file1 file2 directory1
  - To extract a TAR archive: tar -xvf archive.tar
  - To create a compressed TAR archive: tar -czvf archive.tar.gz file1 file2 directory1
  - To extract a compressed TAR archive: tar -xzvf archive.tar.gz
  - To view contents of compressed tar without untarring: tar -ztvf archive.tar.gz
-

ps:

ps aux more informative

- "a": display the processes of all users
- "u": user-oriented format → more details
- "x": list the processes without a controlling terminal
  - started on boot time and running in the background

-
- kill needs pid, -9 kills it without questions.

w

>: Output of a command redirected to a file

- command > file same as command 1> file (stdout redirected, stderr still screen)
- command 2> file (send stderr to file, stdout is screen)
- command 2> error.txt 1> out.txt (send both to different files)
- command > file 2>&1 (send both to same file)
- command 2> /dev/null (suppress error messages)
  - /dev/null is a special file that discards anything written to it

<: stdin of “command” read from “file.txt

- command < file.txt

Pipe takes output from one command and feeds it as input to another command

Command substitution: Output of a command replaces the command itself

- $(command) or
- `command`

Permissions for a file or directory may be any or all of

- r - read; w - write; x - execute
- a directory must have both r and x permissions if the files it contains are to be accessed

Each permission (rwx) can be controlled at three levels:

- u (user = yourself)
- g (group, a set of users)
- o (others, everyone else)

- chmod is used to change the perms of a particular file
- if you want to access a file you must have rx perms
- a - All users, identical to ugo

chmod g=r filename; chmod a-x filename;

- read = 4 write = 2 execute =1

chmod -R 700 dirname (Recursively set read, write, and execute permissions to the file owner and no permissions for all other users on a given directory)

apt is for installing packages.

cut:

1. cut: helps cut parts of a line by delimiter, byte position, and character ("-f" specifies field(s), "-b" specifies bytes(s), "-d" specify a delimiter, --complement complement the selection and --output-delimiter specify a different output delimiter string)
2. cut students.csv -d ',' -f 1,3

- if I put -4 after -f it would rep all fields till 4

diff

- The diff utility compares the contents of file1 and file2 and writes to the standard output the list of changes necessary to convert one file into the other.
- No output is produced if the files are identical.
- Checkout options : -B, -w

```
└─❯ cat test.txt
first
second
third
┌─  ~/Desktop/cs104/tutorials/
└─❯ cat test_repl.txt
fast
third
┌─  ~/Desktop/cs104/tutorials/
└─❯ diff test.txt test_repl.txt
1,2c1
< first
< second
---
> fast
```

Use tail with *(-n +2)*, so as to get rid of first line.

- paste helps combine files horizontally, -d for delimiter, -s for one file at a time.
- Date is a command which tells the current date. With option +%s, it tells the seconds passed since January 1st, 1970 at 00:00:00 UTC
- touch is a command which can be used to create empty files.

- First field: - for File, d for Directory, l for Link
- Second,third,fourth fields: permissions for owner, group and others
- Fifth field: specifies the number of links or directories inside this directory
- Sixth field: user
- Seventh field: group
- Eighth field: size in bytes (use -lh option for better understanding)
- Ninth field: date of last modification
- Tenth field: name of the file/directory

HTML and CSS:

- <hr> tag prints a line like a divider.
- <pre> refers to pre-formatted text.
- <h1 style = “color:blue”> is inline styling.
- <mark> tag highlights text.
- <del> tag strikethroughs text.
- <ins> underlines text.
- <sub> puts the text in subscript
- <sup> puts the text in superscript.
- <link rel=”icon” type = “image/x-icon” href={source}> inserts an image with the source.
- you can use an image tag to insert an image which on clicking leads to a different webstie.
- if target=”_blank” the link is opened in a new window.
- table tag creates a table, tr creates a record and th tells html that it’s a heading.
- if you put td instead, html will take it as a data point.
- ol – ordered list, ul is unordered list. You put elements of the list in li tag.

```
<dl>
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl>
```

Coffee
- black hot drink
Milk
- white cold drink

, Selector p.que describes styling for all <p> tags with class = “que”

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  Your browser does not support the video tag.
</video>
```

for inserting video.

```
<iframe src="demo_iframe.htm" name="iframe_a" height="300px" width="100%"
title="Iframe Example"></iframe>

<p><a href="https://www.w3schools.com" target="iframe_a">W3Schools.com</a></p>
```

what the last tag line does is that when you click the link it puts the website in that box.

<input type="text">	Displays a single-line text input field
<input type="radio">	Displays a radio button (for selecting one of many choices)
<input type="checkbox">	Displays a checkbox (for selecting zero or more of many choices)
<input type="submit">	Displays a submit button (for submitting the form)
<input type="button">	Displays a clickable button

Javascript:

```
<button onclick="document.getElementById('myImage').src='pic_bulboff.gif'">Turn off the
light</button>
```

changing the source of the image.

```
<button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">Click
Me!</button>
```

```
<button type="button" onclick="document.getElementById('demo').style.display='none'">Click
Me!</button>
```

hiding html elements

JavaScript Variables can be declared in 4 ways: Automatically; Using var; Using let; Using const

```
<script>
let x = 5;
let y = 6;
let z = x * y;
```

“==” equal to vs “===” equal value and equal type

No explicit declaration of type! Interpreted based on context!

JavaScript has 8 Datatypes:

- String, Number, Bigint, Boolean, Undefined, Null, Symbol, Object
- Object data type can contain:
  - An object, array or date

Type of a variable can change during execution of the program
Same variable can be used to hold different data types
When adding a number and a string, JavaScript will treat the number as a string
Evaluates expressions from left to right

ARRAY FUNCTIONS:

- length property of an array returns the length of an array
- add a new element to an array using the push() method:
- pop() method removes the last element from an array:
- sort() method sorts an array alphabetically:
- reverse() method reverses the elements in an array.

```
cars.push("Honda");

cars.push("Toyota");

cars.pop();

let size = cars.length;
```

Functions

- Defined with “function” keyword, followed by a name, followed by parentheses ()
- Code to be executed, by the function, is placed inside curly brackets {}

Objects are variables but can contain many values and also methods!

- values are written as name:value pairs (called properties)
- Can be accessed via objectName.propertyName or objectName["propertyName"]

- “this” keyword refers to an object
  - which object depends on how this is being invoked (used or called)?

When a JavaScript variable is declared with the keyword "new", variable is created as an object

```
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
```

USE OF THIS!!!! this is called a method.

USING TRY AND CATHC BLOCKS

- "try" statement allows you to define a block of code to be tested for errors while it is being executed.
- "catch" statement allows you to define a block of code to be executed, if an error occurs in the try block
  - JavaScript statements try and catch come in pairs
- "throw" statement allows you to create a custom error

- Declaring an array constant doesn't mean that you can't change the elements of the array.
- But apde you can't change the array itself, in the sense you can't reference it to a new array.

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
}
```

constructor in java script

Classes

- Classes are not objects, they are just a template for objects!
- Use keyword class to create a class
  - Always add a method named constructor() (same exact name)
    - Executed automatically when a new object is created and is used to initialize object properties

for loops go through the keys of an object, called

```
let text = "";
for (let x of cars) {
  text += x + "<br>";
}
```

Using document.write() after an HTML document is loaded, will delete all existing HTML. Should only be used for testing.

Writing into an HTML element, using innerHTML
Writing into the HTML output using document.write()
Writing into an alert box, using window.alert()
Writing into the browser console, using console.log()

Event Types

UI Events

- Click
- Double click
- Mouseover
- Mouseout
- Keydown, keyup
- Focus, blur

Form Events:

- Submit
- Change
- Input

Document/Window Events:

- Load
- Resize
- Scroll

In a browser window the global object is `[object Window]`:

```
<script>
    document.getElementById("myButton").addEventListener("click",
function() {
        alert("Button clicked!");
    });
</script>
```

the function definition is also an argument.

```
Enter your name: <input type="text" id="fname" oninput="upperCase()">
<p>When you write in the input field, a function is triggered to transform the
input to upper case.</p>

<script>
function upperCase() {
  const x = document.getElementById("fname");
  x.value = x.value.toUpperCase();
}
</script>
```

```
<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;">
Mouse Over Me</div>

<script>
function mOver(obj) {
  obj.innerHTML = "Thank You"
}

function mOut(obj) {
  obj.innerHTML = "Mouse Over Me"
}
</script>
```

its like “hey, if the mouse goes over this div, change the text!!”

```
<script>
const para = document.createElement("p");
const node = document.createTextNode("This is new.");
para.appendChild(node);
const element = document.getElementById("div1");
element.appendChild(para);
</script>
```

Noscript

The HTML <noscript> tag defines an alternate content to be displayed to users that have disabled scripts in their browser or have a browser that doesn't support scripts:

Example:

<noscript>Sorry, your browser does not support JavaScript!</noscript>

The value property is used to get the value of an input element.

the onclick attribute is used to call the function when the button is clicked..

```
<button onclick="calculateSquareRoot()">Calculate Square Root</button>
```

We do not use the link tag to link to external javascript files. We use the self-closing script tag instead.

```
<script src="functions.js"></script>
```

```
document.addEventListener("keydown", function(event){
    document.getElementById("demo").innerHTML = "You pressed the " + event.key + " key.";
});

document.addEventListener("keyup", function(event){
    document.getElementById("demo").innerHTML = "You released the " + event.key + " key.";
})
```

```
<script>
    let box1 = document.getElementById("box1");
    let box2 = document.getElementById("box2");
    let box3 = document.getElementById("box3");
    box1.addEventListener("mouseover", function(){
        box1.style.backgroundColor = "red";
    });

    box1.addEventListener("mouseout", function(){
        box1.style.backgroundColor = "blue";
    });

    box2.addEventListener("mousedown", function(){
        box2.style.backgroundColor = "green";
    });

    box2.addEventListener("mouseup", function(){
        box2.style.backgroundColor = "blue";
    });

    box3.addEventListener("dblclick", function(){
        box3.style.backgroundColor = "yellow";
    });

    box3.addEventListener("click", function(){
        box3.style.backgroundColor = "orange";
    });
```

```
<script>
    window.addEventListener("load", function(){
        alert("The page has loaded.");
    });

    window.addEventListener("scroll", function(){
        document.body.style.backgroundColor = "red";
    });

    window.addEventListener("resize", function(){
        document.body.style.backgroundColor = "blue";
    });

    window.addEventListener("click", function(){
        document.body.style.backgroundColor = "green";
    });
</script>
```

- .reset() for form resetting
- alert() for popping up an alert window.

GDB and Make:

Add output statements to the code

- Good to send debugging information to standard erro
  - Standard output may be redirected to files or other programs (through pipes etc)
  - Comment out when not needed

```
for (int i = 0; i < numElements; ++i)
{
  // cerr << "i: " << i << endl;
   cout << myArray[i] << endl;
}
// cerr << "Exited loop" << endl;
myArray[0] = myArray[1];
```

Too many comments?

- Set a compile-time symbol DEBUG
  - "#define DEBUG 1" in a .h file (or)
  - g++ -g -c -DDEBUG myProgram.cpp

- A way of defensive programming
- A boolean test that should be true if things are working as expected
  - Assert macro is defined in <assert.h> for C and <cassert> for C++
  - Program stops with an informational message whenever the asserted condition fails

If NDEBUG is defined, then each assertion is compiled as a comment

g++ -g -o broken broken.cpp

execution, can use breakpoints

- To specify a breakpoint upon entry to a function
  - (gdb) break function
  - (gdb) break bug.c:function
  - (gdb) b function (shortcut b for break)
- Set a breakpoint by specifying a file and line number
  - (gdb) break 26 (line no)
  - (gdb) break bug.c:26

- (gdb) help

"run" will run the program

- (gdb) run

- To list where the current breakpoints are set
  - (gdb) info breakpoints
- To delete the breakpoint numbered 2 (as specified via info)
  - (gdb) delete 2
- Can also set breakpoints to only trigger when certain conditions are true.
  - (gdb) break 23 if i == count - 1
    (e.g. for (int i = 0; i < count; i++))

- Can check the values of certain key variables at breakpoint
  - (gdb) print n
  - (gdb) p n (shortcut print is p)
- Can also use print to evaluate expressions, make function calls, reassign variables, and more
  - (gdb) print buffer[0] = 'Z'
  - (gdb) print strlen(buffer)

info args prints out the arguments (parameters) to the current function you're in:

- (gdb) info args

info locals prints out the local variables of the current function:

- (gdb) info locals

At a breakpoint, we can ask the debugger to print out the contents of the file via list

- (gdb) list
- We can also list the contents of files by name and line number
  - (gdb) list bug.C:1

Can examine the contents of the stack via backtrace which prints the current functions on the stack

- (gdb) backtrace
- (gdb) bt (bt shortcut for backtrace)
- Also prints the arguments to the functions on the call stack

Profiling

- Debuggers enable us to search for/localize bugs and observe program behaviour
- Profilers allow us to examine the program's overall use of system resources
  - focused primarily on memory use and cpu consumption
  - Can determine parts of code that are time consuming and need to be re-written for faster execution

- Have profiling enabled while compiling the code
  - g++ -pg -o profile profile.cpp
- Execute the program code to produce the profiling data
  - ./profile
  - Generates gmon.out (a binary file)
    - If gmon.out already exists, it will be overwritten.
    - Note program must exit normally (don;t use control-c)
- Run the gprof tool on the profiling data file
  - gprof profile gmon.out > analysis.txt

MAKE:

target: dependencies
command
command
command

- Target: mostly name of a file that is generated by a program

Command: an action that needs to be carrie

- Need to start with a tab character, not spaces

Variables are strings and assigned values via =

- E.g. OBJ = udp-test.o sockutil.o timeutil.o

Can reference variables using either ${} or $()

- $(OBJ)

Phony target: name for some commands to be executed (not name of a file)

- Add -k when running make to continue running even in the face of errors
  - Helpful if you want to see all the errors of Make at once
- Add a - before a command to suppress the error

`GNUmakefile', `makefile' and `Makefile'

- Use a different name, use make -f

g++ -o output_file input_file.cpp:

1. -Wall: Enable most warnings
2. -std=c++20: Use C++20 standard
3. -g: Generate debugging information
4. -O2: Optimize code

```
hey: one two
  # Outputs "hey", since this is the target name
  echo $@

  # Outputs all prerequisites newer than the target
  echo $?

  # Outputs all prerequisites
  echo $^

  touch hey
```

Makefile

```
1    %.o: %.c
2        echo "hello"
```

CC: Program for compiling C programs, default cc
CXX: Program for compiling C++ programs, default g++
CFLAGS: Extra flags to give to the C compiler.
CXXFLAGS: Extra flags to the C++ compiler.
CPPFLAGS: Extra flags to give to the C preprocessor.
LDFLAGS: Extra flags to give to compilers when they are supposed to invoke the linker.

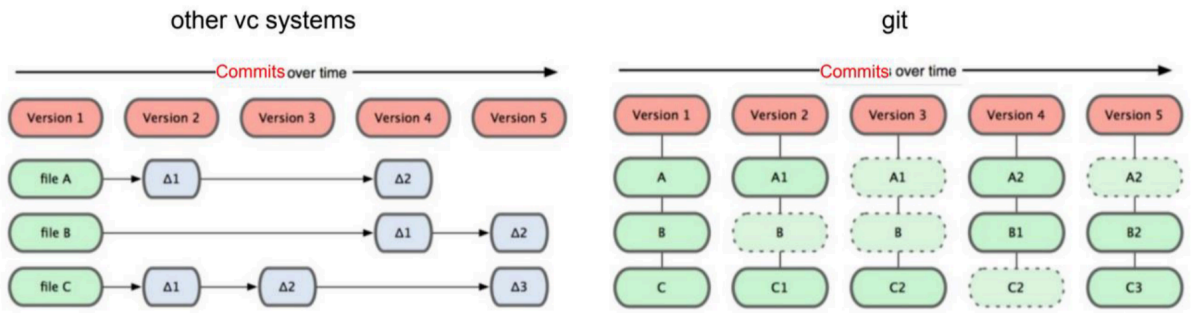
break +offset: will set a breakpoint after offset lines from current spot.
tbreak: temporary break, remove when reached
rbreak regex: break on all functions matching regex
Ignore n count: ignore breakpoint n count times.

backtrace/bt: Prints the current contents of the stack. Also prints the arguments to the function calls
up: Helps move one step up in the stack i.e. from a function to it's caller.
down: Helps move one step down in the stack, i.e. from a function to it's callee.

display <variable>: Print the values of the variable, every time the program stops at a breakpoint.

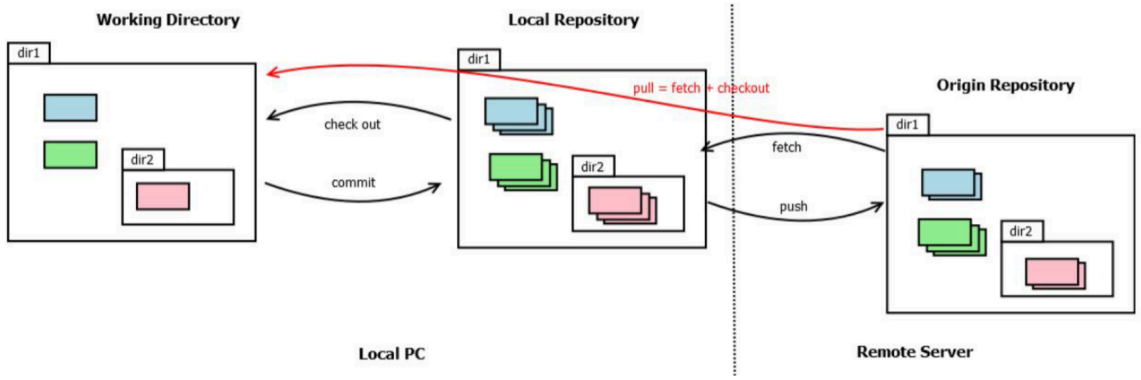
Do not include .h files in the dependencies nor in the compiling statement.

Git:



- Repository: collection of versions of files
  - Tracks deleted and newly added files
  - Users do not edit or even read files in the repo

Checkout: ask repository to give a copy of a version of a file



Fetch: fetch changes anyone else may have made from the origin to our local repository

- Fetching simply updates local repository
- Need checkout for them to reflect in working directory.

Staging: We “add” files to stage and then commit from stage instead of the working directory

- "init" : Used to create a Git repository
  - git init

- Add files to staging area
  - git add file1.txt file2.txt

git commit -a: Commit a snapshot of all changes in the working directory

git commit -m "commit message":

git commit --amend: modifies the last commit
- Instead of creating a new commit, staged changes will be added to the previous commit

git log

git log file1.txt (commit history of that file)

- A long hexadecimal number you see is the commit's hash, helps identify a commit
  - can use just 5 digits mostly in commands

git show :filename
Example: git show :file1.txt
Shows the content of file1.txt in the staging area

git show commit:filename
Example: git show HEAD:file1.txt
Shows the content of file1.txt in HEAD

Example: git show 5b80ea8:file1.txt
Shows the content of file1.txt in the commit object 5b80ea8

git diff <commit>: shows the diff between the current working tree and the <commit>

git diff --cached <commit>: shows the diff between your staged changes and the <commit

- diff a.txt b.txt
  - Output:
    - Line numbers corresponding to the first file
    - A special symbol
    - Line numbers corresponding to the second file
    - E.g. 2,3c3
      - line 2 to line 3 in the first file needs to be changed to match line number 3 in the second file
    - Lines preceded by a < are lines from the first file.
    - Lines preceded by > are lines from the second file.
    - The three dashes ("—") merely separate the lines of file 1 and file 2

diff -u a.txt b.txt (unified mode)

- Output:
  - The first file is indicated by `---`, and the second file is indicated by `+++`.
- Unchanged lines are displayed without any prefix
- Lines in the first file to be deleted are prefixed with -
- Lines in the second file to be added are prefixed with +.

g

- You can move backwards in time by checking out an older commit.
  - git checkout commit-id
  - Will replace the contents of working directory by the contents of that older commit

Get back to most recent commit via git checkout master

Ability to rollback individual files to old versions: git checkout commit-id path-to-a-file

- Then can use git commit if you want everything else to use current and this file to be some older version

HEAD answers the question: “Where am I right now?”

Most of the time, HEAD points to a branch name

- So far we have seen only one branch, master!
- HEAD is synonymous with “the last commit in the current branch.”
  - This is the normal state

In a detached HEAD state; HEAD is pointing directly to a commit instead of a branch

git branch: List the branches

git switch -c testing: create a new branch

- “testing” is the name of this new branch

the folder and then commit using the “-am” flag. With the -a option, Git automatically stages all **tracked files** with changes, so you don't need to use git add. You will notice via the last three commands that file1.txt and file2.txt are v2 now. Note file3 is not committed yet. This is because it is not being tracked. file1/2 were added at some point in the past to staging and are being tracked, hence they were auto-committed via -a. If you want file3 committed, you have to first add it to staging. After this it will be tracked

g

if you make some changes in a file, but don’t commit it, and you try to checkout that, it would return an error.

c. git log

4. Now let us checkout again. Should show file1:v1, and no file2; HEAD is pointing to first commit and is in detached head state! Note master is at the latest commit still. File3 shows still since it is in the folder, but file2 won’t since we did not commit it in the first commit.

Let us introduce conflict. Ensure you are in master. Make some changes in the master branch.

a. git checkout master
b. echo "v5" > file1
c. git commit -am "mainbranch file1:v5 file2:v3
d. cat file1.txt file2.txt

Now let us go to the new branch. And make some changes there as well. Will change file1 to v6 and also change file3 to v2 and commit both

a. git checkout newbranch
b. cat file1.txt file2.txt file3.txt
c. echo "v6" > file1.txt
d. echo "v2" > file3.txt
e. git add file3.txt
f. git commit -am " In newbranch: file1:v6, file2:v3, file3:v2"

Let us merge again in master. Git will fail since there is a conflict when merging.

a. git checkout master
b. cat file1.txt file2.txt file3.txt
c. git merge -m "merging" newbranch

We now have to decide whether file1 should be v5 or v6. Let us say, we want it to be v6, in which case, let us checkout that version into the master (first command) and then commit everything and then merge. Note master still has no file3 but after merge, file3 will come to master branch. There was no conflict to resolve wrt this file.

a. git checkout newbranch file1
b. cat file1.txt file2.txt file3.txt
c. git commit -am "main file1:v6 file2:v3"
d. git merge -m "merging" newbranch
e. ls
f. cat file1.txt file2.txt file3.txt

how to resolve conflicts

- git config tells who did the changes when log is done later.
- git add . □ adds all files in that particular folder
- git add -p □ allows to select a selective part of file
- git checkout -b {branch name} □ creates branch and moves to the branch

bash:

- be careful between difference between single and double quotes.
- $({variable}) dereferences the variable.
- echo ${var}XX${word} makes sure that variable abc xyzXX123 is printed when var = “abc xyz” and word is “123”
- if you don’t use curly braces, then it will think that varxx is a variable, and will return a 0 value.
- echo word word just prints in word word
- echo “word word” prints in word word
- ls “*.sh” it just looks literally
  let here contrary to javascript, let is used to evaluate arithmetic expns
- everything by default is seen as a string
- 0 is treated as a success condition, so in if statements it is evaluates to true.
- while do {looooopy stuff here} done < file.txt its telling dude feed in file.txt into the while loop
- in the while loop, if you use -r it preserves the escape sequences

Tut:

- $# number of command line arguments
- $0 refers to the bash file name, the script name
- {\ } is an escape sequence for putting space in terminal
- $* all arguments in a single string
- $@ all arguments in separate strings
- you can check the difference by using a for loop.
- $? returns status of most recently executed command
- $$ process ID of current process

if [! -e “$filepath”]: then

    echo “Error: file not found”

fi says if doesn’t exist, throw error

- put arithmetic expressions in $(({here}))
- {filename}(){

        function stuff here

  }

- local {variable name} = “1” creates a local variable
- echo statement within a function is like a return statement
- read -p “enter:” y reads a value from terminal and puts it it in y.
- $1, $2, refer to the positional argument given to a function
- files = (‘f1’ ‘f2’ ‘f3’) array declaration
- accessing elements in array ${file[0]} and if you want to print all things ${file[*]}
- declare -A distros
- distros[‘hi’] = ‘hi’

- distros[‘bye’] = ‘bye’
- you access dictionary values in a way similar to lists, but instead of index, you put in the value of the string
- if you put a ! mark before, it gives all keys
- if you don’t you get all values
- let “x++” increases the value of x by 1 {the double quotes btw is not necessary}
- -d asks whether the thing is a directory
- -f asks whether the thing is a file
- -z checks whether the string is empty
- -n checks whether the string is not empty
- string comparision just == is used
- != for not equal to
- for numbers we use -lt -gt -eq -ne
- conditions within square brackets
- && for logical and and || for logical or
- [[{the entire logical expression, you need not break it in between when you have logical ands and ors in between}]]

FOR:

- for variable in list
- do
    - command 1
    - command 2
- done