Bare Bones Bash

Thiseas C. Lamnidis

Aida Andrades Valtueña

Aims of this session

- Aim:
  - Familiarise yourself with basic concepts and commands of bash

Aims of this session

- Aim:
  - Familiarise yourself with basic concepts and commands of bash
- Objectives
  - What is a terminal? What is a command prompt?
  - What is the difference between Absolute and Relative paths?
  - How can you move around the filesystem and interact with files and/or directories?
  - What are data streams, pipes, and redirects?
  - Finding documentation for bash tools.
  - What is a variable?
  - Difference between ' and "!
  - Parameter expansion!!

The Five Commandments of Bare Bones Bash

1. Be lazy!

- Desire for shortcuts motivates you to explore more!

1. Be lazy!

- Desire for shortcuts motivates you to explore more!

2. ~~Google~~ The Hive-Mind knows everything.

- 99% of the time, someone else has already had the same issue.

1. Be lazy!

- Desire for shortcuts motivates you to explore more!

2. ~~Google~~ The Hive-Mind knows everything.

- 99% of the time, someone else has already had the same issue.

3. Document everything you do.

- Make future you happy

1. Be lazy!

- Desire for shortcuts motivates you to explore more!

2. ~~Google~~ The Hive-Mind knows everything.

- 99% of the time, someone else has already had the same issue.

3. Document everything you do.

- Make future you happy

4. There will ALWAYS be a typo!

- Don't get disheartened, even best programmers make mistakes

1. Be lazy!

- Desire for shortcuts motivates you to explore more!

2. ~~Google~~ The Hive-Mind knows everything.

- 99% of the time, someone else has already had the same issue.

3. Document everything you do.

- Make future you happy

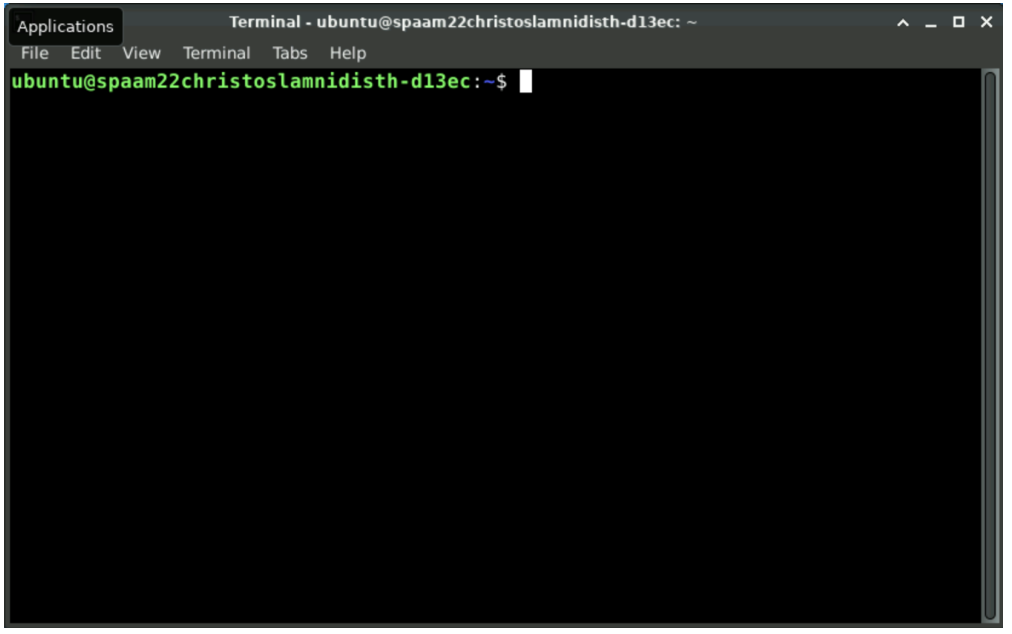
4. There will ALWAYS be a typo!

- Don't get disheartened, even best programmers make mistakes

5. Don't be afraid of you freedom!

- Explore! Try out things!

Preparation!


Applications
Terminal - ubuntu@spaam22christoslamnidisth-d13ec: ~
File Edit View Terminal Tabs Help
ubuntu@spaam22christoslamnidisth-d13ec:~$


$BBB

Always mind the $ and >!

Absolute and Relative paths

In addition to your command prompt, you can use `pwd` to see your current directory

```
$ pwd
```

Absolute and Relative paths

In addition to your command prompt, you can use `pwd` to see your current directory

```
$ pwd
```

```
/home/ubuntu
```

Absolute and Relative paths

In addition to your command prompt, you can use `pwd` to see your current directory

```
$ pwd
```

```
/home/ubuntu
```

- ~ is a **relative** path, while `pwd` returns an **absolute** path

Let's talk about paths!

You have just arrived to Leipzig for a summer school that is taking place at MPI-EVA. After some questionable navigation, you find yourself at the Bayerische Bahnhof. Tired and disheartened, you decide to ask a local.

Let's talk about paths!

You have just arrived to Leipzig for a summer school that is taking place at MPI-EVA. After some questionable navigation, you find yourself at the Bayerische Bahnhof. Tired and disheartened, you decide to ask a local.

You see a friendly-looking metalhead, and decide to ask them for directions!

Let's talk about paths!

You have just arrived to Leipzig for a summer school that is taking place at MPI-EVA. After some questionable navigation, you find yourself at the Bayerische Bahnhof. Tired and disheartened, you decide to ask a local.

You see a friendly-looking metalhead, and decide to ask them for directions!

A friendly-looking metalhead.

Happy to help, but I only use **absolute paths.**
From Leipzig Hbf, take Querstraße southward.
Continue straight and take Nürnberger Str. southward until you reach Str. des 18 Oktober.
Finally take Str. des 18 Oktober. moving southeast until you reach EVA!

Let's talk about paths!

You have just arrived to Leipzig for a summer school that is taking place at MPI-EVA. After some questionable navigation, you find yourself at the Bayerische Bahnhof. Tired and disheartened, you decide to ask a local.

You see a friendly-looking metalhead, and decide to ask them for directions!

A friendly-looking metalhead.

Happy to help, but I only use **absolute paths.**
From Leipzig Hbf, take Querstraße southward.
Continue straight and take Nürnberger Str. southward until you reach Str. des 18 Oktober.
Finally take Str. des 18 Oktober. moving southeast until you reach EVA!

Examples of absolute paths:

```
/home/ubuntu
/Hbf/Querstraße/Nürnberger_Str/Str_18_Oktober/Deutscher_Platz/EVA
```

Let's talk about paths!

Not sure how to get back to Leipzig Hbf to apply those directions, you decide to ask someone else for directions.

Let's talk about paths!

Not sure how to get back to Leipzig Hbf to apply those directions, you decide to ask someone else for directions.

This street is Str. des 18 Oktober. Walk straight that way till you walk past the tram tracks and you will reach EVA!

A friendly-looking local.

Let's talk about paths!

Not sure how to get back to Leipzig Hbf to apply those directions, you decide to ask someone else for directions.

This street is Str. des 18 Oktober. Walk straight that way till you walk past the tram tracks and you will reach EVA!

A friendly-looking local.

Examples of relative paths:

~

```
./Str_18_Oktober/Deutscher_Platz/EVA
```

The different types of file paths

Absolute

- The location of a file or folder, **from the "root" directory** (/).

Relative

- The location of a file or folder, **from your current directory.**

The different types of file paths

Absolute

- The location of a file or folder, **from the "root" directory** (/).

Relative

- The location of a file or folder, **from your current directory.**

When writing code it is better to use **absolute** paths, so your code works independently of the users's current directory!

Basic bash commands

- **List** directory contents:

```
$ ls
```

```
Desktop    Downloads    'MEGA X'    Pictures    Templates    bin
Documents  M11CC_Out    Music       Public      Videos       thinclient_drives
```

Basic bash commands

- **List** directory contents:

```
$ ls
```

- **Make** a directory:

```
$ mkdir barebonesbash
```

Basic bash commands

- **List** directory contents:

```
$ ls
```

- **Make** a directory:

```
$ mkdir barebonesbash
```

- **Move** (or rename) files and directories

```
$ mv barebonesbash BareBonesBash
```

Basic bash commands

- **List** directory contents:

```
$ ls
```

- **Make** a directory:

```
$ mkdir barebonesbash
```

- **Move** (or rename) files and directories

```
$ mv barebonesbash BareBonesBash
```

- **Change** directories

```
$ cd BareBonesBash
```

Basic bash commands

- **Download** a remote file to your computer

```
$ wget git.io/Boosted-BBB-meta
```

Basic bash commands

- **Download** a remote file to your computer

```
$ wget git.io/Boosted-BBB-meta
```

- **Copy** a file or directory to a new location

```
$ cp Boosted-BBB-meta Boosted-BBB-meta.tsv
```

Basic bash commands

- **Download** a remote file to your computer

```
$ wget git.io/Boosted-BBB-meta
```

- **Copy** a file or directory to a new location

```
$ cp Boosted-BBB-meta Boosted-BBB-meta.tsv
```

- **Remove** (delete) files

```
rm Boosted-BBB-meta
```

Basic bash commands

- Con**cat**enate file contents to screen

```
$ cat Boosted-BBB-meta.tsv
```

Basic bash commands

- Con**cat**enate file contents to screen

```
$ cat Boosted-BBB-meta.tsv
```

- See only the **first**/**last** 10 lines of a file

```
$ head -n 10 Boosted-BBB-meta.tsv
$ tail -n 10 Boosted-BBB-meta.tsv
```

Basic bash commands

- Con**cat**enate file contents to screen

```
$ cat Boosted-BBB-meta.tsv
```

- See only the **first**/**last** 10 lines of a file

```
$ head -n 10 Boosted-BBB-meta.tsv
$ tail -n 10 Boosted-BBB-meta.tsv
```

- Look at the contents of a file **interactively** (quit with `q`)

```
$ less Boosted-BBB-meta.tsv
```

Basic bash commands

- Con**cat**enate file contents to screen

```
$ cat Boosted-BBB-meta.tsv
```

- See only the **first**/**last** 10 lines of a file

```
$ head -n 10 Boosted-BBB-meta.tsv
$ tail -n 10 Boosted-BBB-meta.tsv
```

- Look at the contents of a file **interactively** (quit with `q`)

```
$ less Boosted-BBB-meta.tsv
```

- **Count** the number of **l**ines in a file

```
$ wc -l Boosted-BBB-meta.tsv
```

Datastreams,

Piping,

and redirects

Datastreams

Programs can take in and spit out data from different *streams*. By default there are 3 such data streams.

Datastreams

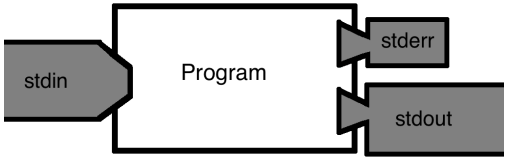
Programs can take in and spit out data from different *streams*. By default there are 3 such data streams.

- `stdin`: the **st**an**d**ard **in**put
- `stdout`: the **st**an**d**ard **out**put
- `stderr`: the **st**an**d**ard **err**or

Datastreams

Programs can take in and spit out data from different *streams*. By default there are 3 such data streams.

- `stdin`: the **st**and**d**ard **in**put
- `stdout`: the **st**and**d**ard **out**put
- `stderr`: the **st**and**d**ard **err**or



Piping

Piping lets you combine commands together using |

```
$ head -n 10 Boosted-BBB-meta.tsv | wc -l
```

Piping

Piping lets you combine commands together using |

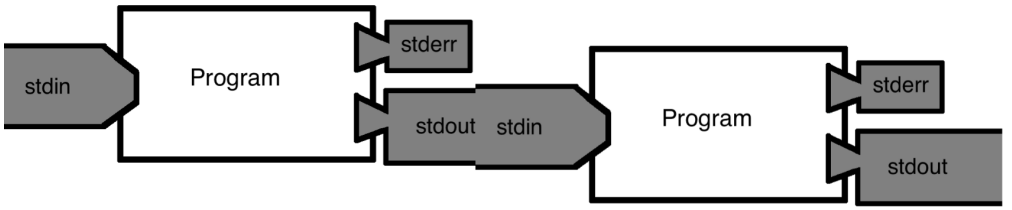
```
$ head -n 10 Boosted-BBB-meta.tsv | wc -l
```

```
10
```

Piping

Piping lets you combine commands together using `|`

```
$ head -n 10 Boosted-BBB-meta.tsv | wc -l
```



The `stdout` of one script becomes the `stdin` of the other. `stderr` is always printed on your screen.

Redirects

Much like streams in the real world, datastreams can be redirected.

Redirects

Much like streams in the real world, datastreams can be redirected.

- `stdin` can be redirected with `<`.

Redirects

Much like streams in the real world, datastreams can be redirected.

- `stdin` can be redirected with `<`.
- `stdout` can be redirected with `>`.

Redirects

Much like streams in the real world, datastreams can be redirected.

- `stdin` can be redirected with `<`.
- `stdout` can be redirected with `>`.
- `stderr` can be redirected with `2>`.

Redirects

Much like streams in the real world, datastreams can be redirected.

- `stdin` can be redirected with `<`.
- `stdout` can be redirected with `>`.
- `stderr` can be redirected with `2>`.

```
$ head -n 10 Boosted-BBB-meta.tsv | wc -l >linecount.txt
$ cat linecount.txt
```

Redirects

Much like streams in the real world, datastreams can be redirected.

- `stdin` can be redirected with `<`.
- `stdout` can be redirected with `>`.
- `stderr` can be redirected with `2>`.

```
$ head -n 10 Boosted-BBB-meta.tsv | wc -l >linecount.txt
$ cat linecount.txt
```

```
10
```

Redirects

Much like streams in the real world, datastreams can be redirected.

- `stdin` can be redirected with `<`.
- `stdout` can be redirected with `>`.
- `stderr` can be redirected with `2>`.

```
$ head -n 10 Boosted-BBB-meta.tsv | wc -l >linecount.txt
$ cat linecount.txt
```

```
10
```

You can then **remove** the file we just made

```
$ rm linecount.txt
```

Finding the help you need

You don't always have to google for documentation! Many programs come with in-built helptext, or access to online manuals right from your terminal!

Finding the help you need

You don't always have to google for documentation! Many programs come with in-built helptext, or access to online manuals right from your terminal!

- You can get a **one sentence summary** of what a tool does with `whatis`

```
$ whatis cat
```

```
cat(1)                  - concatenate files and print on the standard output
```

Finding the help you need

You don't always have to google for documentation! Many programs come with in-built helptext, or access to online manuals right from your terminal!

- You can get a **one sentence summary** of what a tool does with `whatis`

```
$ whatis cat
```

```
cat(1)                  - concatenate files and print on the standard output
```

- While `man` gives you **access to online manuals** for each tool (exit with `q`)

```
$ man cat
```

Finding the help you need

You don't always have to google for documentation! Many programs come with in-built helptext, or access to online manuals right from your terminal!

- You can get a **one sentence summary** of what a tool does with `whatis`

```
$ whatis cat
```

```
cat(1)                  - concatenate files and print on the standard output
```

- While `man` gives you **access to online manuals** for each tool (exit with `q`)

```
$ man cat
```

> ***Activity***: *What flag should you give* `cat` *to include line numbers in the output?*

Variables

Variables

A named container whose contents you can expand at will or change.

Variables

A named container whose contents you can expand at will or change.

You can assign variables with = and pull their contents with $

Variables

A named container whose contents you can expand at will or change.

You can assign variables with `=` and pull their contents with `$`

The easiest way to see the contents of a varable is using `echo`!

```
$ echo "This is my home directory: $HOME"
```

Variables

A named container whose contents you can expand at will or change.

You can assign variables with `=` and pull their contents with `$`

The easiest way to see the contents of a varable is using `echo`!

```
$ echo "This is my home directory: $HOME"
```

```
This is my home directory: /home/ubuntu
```

Variables

A named container whose contents you can expand at will or change.

You can assign variables with `=` and pull their contents with `$`

The easiest way to see the contents of a varable is using `echo`!

```
$ echo "This is my home directory: $HOME"
```

```
This is my home directory: /home/ubuntu
```

And now for a trip...

Variables

```
$ GreekFood=4            #Here, 'GreekFood' is a number.
$ echo "Greek food is $GreekFood people who want to know what heaven tastes like."
```

Variables

```
$ GreekFood=4            #Here, 'GreekFood' is a number.
$ echo "Greek food is $GreekFood people who want to know what heaven tastes like."
```

```
Greek food is 4 people who want to know what heaven tastes like.
```

Variables

```
$ GreekFood=4            #Here, 'GreekFood' is a number.
$ echo "Greek food is $GreekFood people who want to know what heaven tastes like."
```

```
Greek food is 4 people who want to know what heaven tastes like.
```

```
$ GreekFood=delicious   #We overwrite that number with a word (i.e. a 'string').
$ echo "Everyone says that Greek food is $GreekFood."
```

Variables

```
$ GreekFood=4            #Here, 'GreekFood' is a number.
$ echo "Greek food is $GreekFood people who want to know what heaven tastes like."
```

```
Greek food is 4 people who want to know what heaven tastes like.
```

```
$ GreekFood=delicious   #We overwrite that number with a word (i.e. a 'string').
$ echo "Everyone says that Greek food is $GreekFood."
```

```
Everyone says that Greek food is delicious.
```

Variables

```
$ GreekFood=4            #Here, 'GreekFood' is a number.
$ echo "Greek food is $GreekFood people who want to know what heaven tastes like."
```

```
Greek food is 4 people who want to know what heaven tastes like.
```

```
$ GreekFood=delicious   #We overwrite that number with a word (i.e. a 'string').
$ echo "Everyone says that Greek food is $GreekFood."
```

```
Everyone says that Greek food is delicious.
```

```
$ GreekFood="Greek wine" #We can overwrite 'GreekFood' again,
## but when there is a space in our string, we need quotations.
$ echo "The only thing better than Greek food is $GreekFood!"
```

Variables

```
$ GreekFood=4            #Here, 'GreekFood' is a number.
$ echo "Greek food is $GreekFood people who want to know what heaven tastes like."
```

```
Greek food is 4 people who want to know what heaven tastes like.
```

```
$ GreekFood=delicious   #We overwrite that number with a word (i.e. a 'string').
$ echo "Everyone says that Greek food is $GreekFood."
```

```
Everyone says that Greek food is delicious.
```

```
$ GreekFood="Greek wine" #We can overwrite 'GreekFood' again,
## but when there is a space in our string, we need quotations.
$ echo "The only thing better than Greek food is $GreekFood!"
```

```
The only thing better than Greek food is Greek wine!
```

Variables

```
$ GreekFood=4            #Here, 'GreekFood' is a number.
$ echo "Greek food is $GreekFood people who want to know what heaven tastes like."
```

```
Greek food is 4 people who want to know what heaven tastes like.
```

```
$ GreekFood=delicious   #We overwrite that number with a word (i.e. a 'string').
$ echo "Everyone says that Greek food is $GreekFood."
```

```
Everyone says that Greek food is delicious.
```

```
$ GreekFood="Greek wine" #We can overwrite 'GreekFood' again,
## but when there is a space in our string, we need quotations.
$ echo "The only thing better than Greek food is $GreekFood!"
```

```
The only thing better than Greek food is Greek wine!
```

```
$ GreekFood=7 #And, of course, we can overwrite with a number again too.
$ echo "I have been to Greece $GreekFood times already this year, for the food and
```

Variables

```
$ GreekFood=4            #Here, 'GreekFood' is a number.
$ echo "Greek food is $GreekFood people who want to know what heaven tastes like."
```

```
Greek food is 4 people who want to know what heaven tastes like.
```

```
$ GreekFood=delicious   #We overwrite that number with a word (i.e. a 'string').
$ echo "Everyone says that Greek food is $GreekFood."
```

```
Everyone says that Greek food is delicious.
```

```
$ GreekFood="Greek wine" #We can overwrite 'GreekFood' again,
## but when there is a space in our string, we need quotations.
$ echo "The only thing better than Greek food is $GreekFood!"
```

```
The only thing better than Greek food is Greek wine!
```

```
$ GreekFood=7 #And, of course, we can overwrite with a number again too.
$ echo "I have been to Greece $GreekFood times already this year, for the food and
```

```
I have been to Greece 7 times already this year, for the food and wine!
```

Quotes matter!

In bash, there is a big difference between a single quote ' and a double quote "!

- The contents of single quotes, are passed on as they are.
- Inside double quotes, contents are *interpreted*!

Quotes matter!

In bash, there is a big difference between a single quote ' and a double quote "!

- The contents of single quotes, are passed on as they are.
- Inside double quotes, contents are *interpreted*!

In some cases the difference doesn't matter:

```
$ echo "I like Greek Food"
$ echo 'I like Greek Food'
```

```
I like Greek Food
I like Greek Food
```

Quotes matter!

In bash, there is a big difference between a single quote `'` and a double quote `"`!

- The contents of single quotes, are passed on as they are.
- Inside double quotes, contents are *interpreted*!

In some cases the difference doesn't matter:

```
$ echo "I like Greek Food"
$ echo 'I like Greek Food'
```

```
I like Greek Food
I like Greek Food
```

In other cases it makes all the difference:

```
$ Arr=Banana
$ echo 'Pirates say $Arr'
$ echo "Minions say $Arr"
```

Quotes matter!

In bash, there is a big difference between a single quote `'` and a double quote `"`!

- The contents of single quotes, are passed on as they are.
- Inside double quotes, contents are *interpreted*!

In some cases the difference doesn't matter:

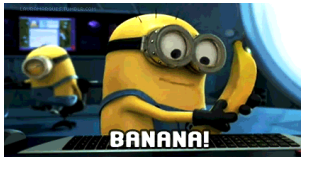
```
$ echo "I like Greek Food"
$ echo 'I like Greek Food'
```

```
I like Greek Food
I like Greek Food
```

In other cases it makes all the difference:

```
$ Arr=Banana
$ echo 'Pirates say $Arr'
$ echo "Minions say $Arr"
```

```
Pirates say $Arr
Minions say Banana
```



Parameter expansion

The basics

Here's an example variable:

```
$ foo="/home/thiseas/folder/subfolder/BBB.is.bae.txt"
```

The basics

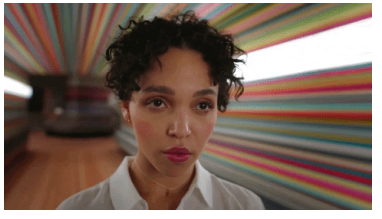
Here's an example variable:

```
$ foo="/home/thiseas/folder/subfolder/BBB.is.bae.txt"
```

To expand a variable use `${}`.

```
$ echo ${foo}
```

```
/home/thiseas/folder/subfolder/BBB.is.bae.txt
```



The basics

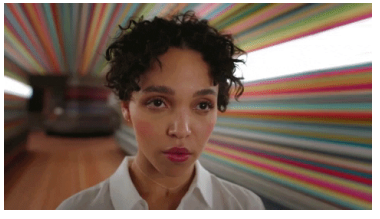
Here's an example variable:

```
$ foo="/home/thiseas/folder/subfolder/BBB.is.bae.txt"
```

To expand a variable use `${}`.

```
$ echo ${foo}
```

```
/home/thiseas/folder/subfolder/BBB.is.bae.txt
```



You can also add a **parameter** to expansions:

```
$ echo ${foo#/home/}
$ echo ${foo#*/}
```

```
thiseas/folder/subfolder/BBB.is.bae.txt
home/thiseas/folder/subfolder/BBB.is.bae.txt
```

Some parameters for expansion

```
$ foo="/home/thiseas/folder/subfolder/BBB.is.bae.txt"
$ echo ${foo}    # No parameters in this expansion
$ echo ${foo#*/} # Removes everything before the first '/'
$ echo ${foo%.*} # What will this do?
```

Some parameters for expansion

```
$ foo="/home/thiseas/folder/subfolder/BBB.is.bae.txt"
$ echo ${foo}    # No parameters in this expansion
$ echo ${foo#*/} # Removes everything before the first '/'
$ echo ${foo%.*} # What will this do?
```

```
/home/thiseas/folder/subfolder/BBB.is.bae.txt
 home/thiseas/folder/subfolder/BBB.is.bae.txt
/home/thiseas/folder/subfolder/BBB.is.bae
```

Some parameters for expansion

```
$ foo="/home/thiseas/folder/subfolder/BBB.is.bae.txt"
$ echo ${foo}    # No parameters in this expansion
$ echo ${foo#*/} # Removes everything before the first '/'
$ echo ${foo%.*} # What will this do?
```

```
/home/thiseas/folder/subfolder/BBB.is.bae.txt
 home/thiseas/folder/subfolder/BBB.is.bae.txt
/home/thiseas/folder/subfolder/BBB.is.bae
```

These expansion can be generalised:

```
$ echo ${foo##*/} # Removes everything before any '/'
$ echo ${foo%%.*} # Removes everything after any '.'
```

Some parameters for expansion

```
$ foo="/home/thiseas/folder/subfolder/BBB.is.bae.txt"
$ echo ${foo}    # No parameters in this expansion
$ echo ${foo#*/} # Removes everything before the first '/'
$ echo ${foo%.*} # What will this do?
```

```
/home/thiseas/folder/subfolder/BBB.is.bae.txt
 home/thiseas/folder/subfolder/BBB.is.bae.txt
/home/thiseas/folder/subfolder/BBB.is.bae
```

These expansion can be generalised:

```
$ echo ${foo##*/} # Removes everything before any '/'
$ echo ${foo%%.*} # Removes everything after any '.'
```

```
BBB.is.bae.txt
/home/thiseas/folder/subfolder/BBB
```

More parameters for expansion

You can use two / to substitute parts of the variable:

```
$ foo="/home/thiseas/folder/subfolder/BBB.is.bae.txt"
$ echo ${foo}                    # No parameters
$ echo ${foo/BBB/BareBonesBash}  # Change BBB to BareBonesBash
```

```
/home/thiseas/folder/subfolder/BBB.is.bae.txt
/home/thiseas/folder/subfolder/BareBonesBash.is.bae.txt
```

More parameters for expansion

You can use two / to substitute parts of the variable:

```
$ foo="/home/thiseas/folder/subfolder/BBB.is.bae.txt"
$ echo ${foo}                    # No parameters
$ echo ${foo/BBB/BareBonesBash}  # Change BBB to BareBonesBash
```

```
/home/thiseas/folder/subfolder/BBB.is.bae.txt
/home/thiseas/folder/subfolder/BareBonesBash.is.bae.txt
```

Leaving the second / out replaces the pattern with "an empty string".

```
$ echo ${foo/BBB}  # Remove BBB
```

More parameters for expansion

You can use two / to substitute parts of the variable:

```
$ foo="/home/thiseas/folder/subfolder/BBB.is.bae.txt"
$ echo ${foo}                    # No parameters
$ echo ${foo/BBB/BareBonesBash}  # Change BBB to BareBonesBash
```

```
/home/thiseas/folder/subfolder/BBB.is.bae.txt
/home/thiseas/folder/subfolder/BareBonesBash.is.bae.txt
```

Leaving the second / out replaces the pattern with "an empty string".

```
$ echo ${foo/BBB}  # Remove BBB
```

```
/home/thiseas/folder/subfolder/.is.bae.txt
```

The last parameter, I swear!

Finally, you can check the length of a variable by using a # BEFORE the variable name.

```
$ foo="/home/thiseas/folder/subfolder/BBB.is.bae.txt"
$ echo ${#foo}  # The length of the variable contents
```

The last parameter, I swear!

Finally, you can check the length of a variable by using a # BEFORE the variable name.

```
$ foo="/home/thiseas/folder/subfolder/BBB.is.bae.txt"
$ echo ${#foo}  # The length of the variable contents
```

```
45
```

So the filepath in `foo` is 45 characters long!

The last parameter, I swear!

Finally, you can check the length of a variable by using a # BEFORE the variable name.

```
$ foo="/home/thiseas/folder/subfolder/BBB.is.bae.txt"
$ echo ${#foo}  # The length of the variable contents
```

```
45
```

So the filepath in `foo` is 45 characters long!

This parameter is more useful when dealing with **bash arrays** (i.e. lists of things).

Recap

You should now understand:

- The difference between the Terminal and the command prompt.
    - What information the command prompt includes.

Recap

You should now understand:

- The difference between the Terminal and the command prompt.
  - What information the command prompt includes.
- The difference between absolute and relative file paths.

Recap

You should now understand:

- The difference between the Terminal and the command prompt.
  - What information the command prompt includes.
- The difference between absolute and relative file paths.
- What data streams are and how to redirect them into files.

Recap

You should now understand:

- The difference between the Terminal and the command prompt.
  - What information the command prompt includes.
- The difference between absolute and relative file paths.
- What data streams are and how to redirect them into files.
- How piping works in bash.

Recap

You should now understand:

- The difference between the Terminal and the command prompt.
  - What information the command prompt includes.
- The difference between absolute and relative file paths.
- What data streams are and how to redirect them into files.
- How piping works in bash.
- How to quickly find documentation about the tools you are using.

Recap

You should now understand:

- The difference between the Terminal and the command prompt.
  - What information the command prompt includes.
- The difference between absolute and relative file paths.
- What data streams are and how to redirect them into files.
- How piping works in bash.
- How to quickly find documentation about the tools you are using.
- What a variable is.
  - How to assign them and expand them.

Recap

You should now understand:

- The difference between the Terminal and the command prompt.
  - What information the command prompt includes.
- The difference between absolute and relative file paths.
- What data streams are and how to redirect them into files.
- How piping works in bash.
- How to quickly find documentation about the tools you are using.
- What a variable is.
  - How to assign them and expand them.
- The difference between single and double quotes in bash.

Recap

You should now understand:

- The difference between the Terminal and the command prompt.
  - What information the command prompt includes.
- The difference between absolute and relative file paths.
- What data streams are and how to redirect them into files.
- How piping works in bash.
- How to quickly find documentation about the tools you are using.
- What a variable is.
  - How to assign them and expand them.
- The difference between single and double quotes in bash.
- How you can use parameters to manipulate variable expansion on the fly!

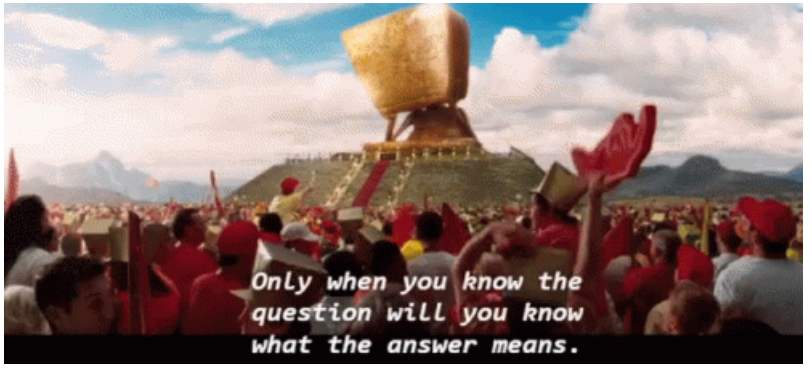
Recap

You should now understand:

- The difference between the Terminal and the command prompt.
  - What information the command prompt includes.
- The difference between absolute and relative file paths.
- What data streams are and how to redirect them into files.
- How piping works in bash.
- How to quickly find documentation about the tools you are using.
- What a variable is.
  - How to assign them and expand them.
- The difference between single and double quotes in bash.
- How you can use parameters to manipulate variable expansion on the fly!

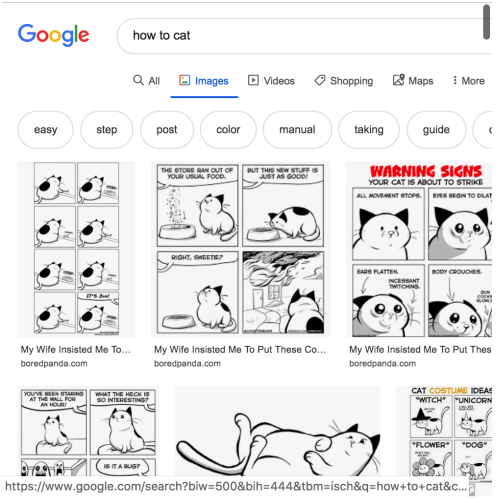
In the next session we will apply some of these concepts together with some new commands to clean up a messy file system.

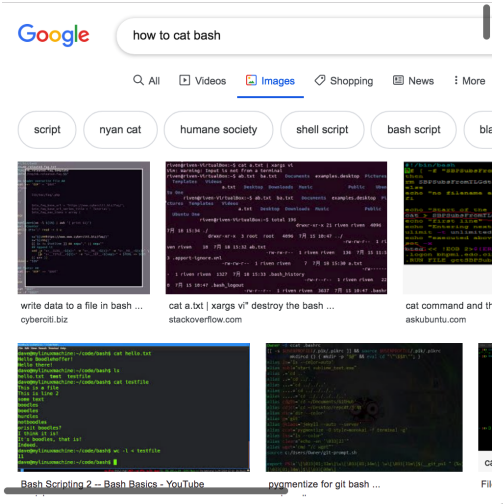
Accessing ~~Google~~ the hive-mind



Knowing the question

- ALWAYS include the name of the language in your query.





Knowing the question

- ALWAYS include the name of the language in your query.
- Broaden your question.

"Hey Google! How to set **X** to **4** in bash?"

"Hey Google! How to set a **variable** to an **integer** in bash?"

Knowing the question

- ALWAYS include the name of the language in your query.
- Broaden your question.
- When you are more familiar, use fancy programmer lingo to make google think you know what you are talking about.

All the cool hackers say:

- "*string*" and not "*text*".
- "*float*" and not "*decimal*".
- Some of these terms can be language specific.