Introduction to Unix

University of Massachusetts Medical School

February 4, 2014

DISCLAIMER

For the sake of clarity, the concepts mentioned in these slides have been simplified significantly.

Most of the command line tools and programs we explain have many more features that can't be found here. Unix command line tools are very well documented and their manual pages are freely available.

These notes are prepared for the users of MGHPCC. MGHPCC system has features that don't exist in standard Unix systems such as `loading modules`.

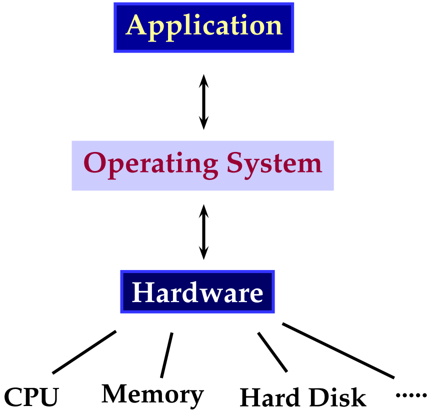
A rigorous treatment of topics on Unix and Bash can be found in various books in several levels.

Contents

- The **Unix** family
- Bash
- Browsing, creating and changing directories
- File operations
- Viewing and processing text files
- Compressed files
- Processes
- Customizing bash

Operating System

An operating system is a collection of software that manages system resources (such as memory, cpu(s), display and etc.) and provide applications a simpler interface to the system hardware.


Application
Operating System
Hardware
CPU
Memory
Hard Disk
.....

Unix

Developed in the 1960’s by Ken Thompson, Dennis Ritchie. It became very popular in the 70’s and 80’s.

"Contrary to popular belief, Unix is user friendly. It just happens to be very selective about who it decides to make friends with."
–anonymous

Unix-like operating systems

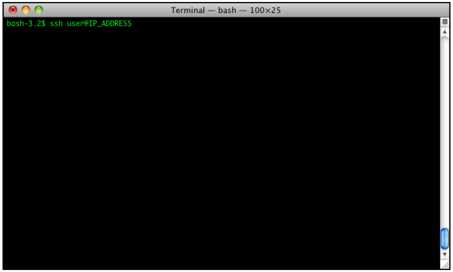
Unix-like operating systems *"behave"* like the original Unix operating system and comply (at least partially) with POSIX (portable operating system interface) standards.

Examples: Linux, OS X, Free BSD, Solaris, Hurd.

Unix $\equiv$ Unix-like $\equiv$ Unix-clone

Welcome to the Unix Family

What is a Terminal?



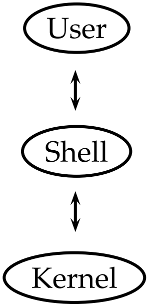
```
bash-3.2$ ssh user@IP_ADDRESS
```

A terminal is a software that emulates a teletype writer terminal used in the early days of Unix.

Unix Shell

A shell is a software that runs inside a terminal and interprets and executes user commands.

One of the most popular shells being used today is called **BASH** (Bourne Again Shell).



Convention

$ denotes the Bash command line prompt. It is not menat to be typed. All Bash commands will appear in a gray box.

```
$ bashcommand
```

Hello Bash

We connect to MGHPCC server.

```
$ ssh username@ghpcc06.umassrc.org
```

Let's verify that we are at the right place.

```
$ hostname
```

Hello Bash

We connect to MGHPCC server.

```
$ ssh username@ghpcc06.umassrc.org
```

Let's verify that we are at the right place.

```
$ hostname
```

BASH Variables

To print something on the screen, we use echo

```
$ echo Hello World
```

This is useful to see the Bash variables!

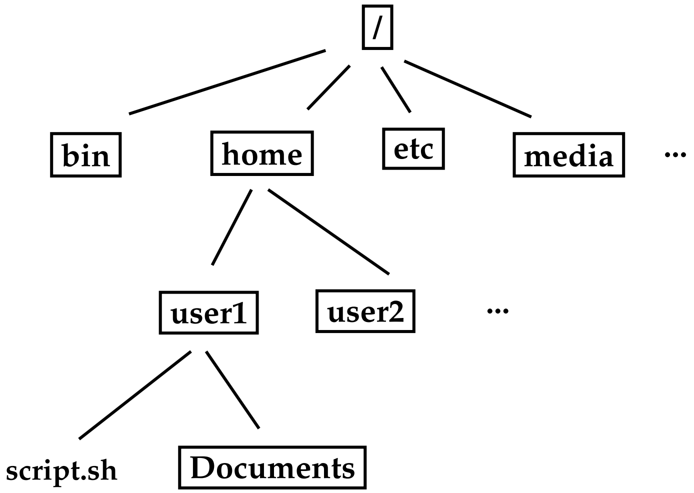
Bash Variables

- Bash variables hold values that are either used by Bash itself or by other programs run by the user.
- To get the value of a Bash variable, we put `$` in the beginning of the variable.
- To assign a value to a variable, we use the syntax `VARIABLENAME=VALUE`

Example:
Let's see our path variable:

```
$ echo $PATH
```

Unix Directory Structure



Bash always have a working directory. To see the working directory,

```
$ pwd
```

To see the contents of the working directory,

```
$ ls
```

To see the contents of a particular directory, provide the path of the directory.

```
$ ls /bin
```

displays th contents of the directory `/bin`

Bash always have a working directory. To see the working directory,

```
$ pwd
```

To see the contents of the working directory,

```
$ ls
```

To see the contents of a particular directory, provide the path of the directory.

```
$ ls /bin
```

displays th contents of the directory `/bin`

Bash always have a working directory. To see the working directory,

```
$ pwd
```

To see the contents of the working directory,

```
$ ls
```

To see the contents of a particular directory, provide the path of the directory.

```
$ ls /bin
```

displays th contents of the directory `/bin`

Command Line Arguments

Usually, we need to provide input to the shell commands or programs.

$$\$\ \ \underbrace{\texttt{ls}}_{\text{executable}}\ \underbrace{\texttt{-lh}}_{\text{argument 1}}\ \underbrace{\texttt{/bin}}_{\text{argument 2}}$$

More on ls

We can get more information about the files or directories by running `ls` with additional arguments.

Compare

```
$ ls
```

with

```
$ ls -l
```

A closer look at ls

`ls` list the given directory contents.
To see the complete list of options,

```
$ man ls
```

This will display the manual page of `ls`.

Some commonly used options are:

-l	:	list contents in more detail
-A	:	list all files in the direcory
-h	:	when used with the `-l` option, prints file sizes in KB, MB, GB, and etc. to reduce the number of digits on the screen.

These options can be grouped

```
$ ls -lh
```

A closer look at ls

`ls` list the given directory contents.
To see the complete list of options,

```
$ man ls
```

This will display the manual page of `ls`.

Some commonly used options are:

-l	:	list contents in more detail
-A	:	list all files in the direcory
-h	:	when used with the `-l` option, prints file sizes in KB, MB, GB, and etc. to reduce the number of digits on the screen.

These options can be grouped

```
$ ls -lh
```

Changing the working directory

First, we see the working directory

```
$ pwd
```

Now, lets change the directory to the `bootcamp` directory.

```
$ cd ~/bootcamp
```

We can verify the change by

```
$ pwd
```

Creating Directories

```
$ mkdir directory_path
```

Note that

```
$ mkdir /home/username/bootcamp/my_new_directory
```

is the same as

```
$ mkdir ~/bootcamp/my_new_directory
```

Copying Files

```
$ cp sourcefile(s) destination_path
```

Let's copy our sample files into the home folder.

```
$ cp ~/bootcamp/transcriptomics.tar.gz ~/transcriptomics.tar.gz
```

Copying Directories

```
$ cp source_directory destination_directory
```

will not work in general.
You need to provide the **-r** parameter.

```
$ cp -r source_directory destination_directory
```

Another useful option is −v which prints the files being copied.

```
$ cp -rv source_directory destination_directory
```

Moving Files

```
$ mv source_file destination_file
```

Deleting Files

`rm` : Remove comand

```
$ rm file_path
```

To remove non-empty directories, use rm with the `-r` option.

```
$ rm -r directory
```

rm is a dangerous command

There is no undoing of rm!
You may want to use it in the ineractive mode not to delete accidentally files

```
$ rm -i file
```

File Path

~	:	Home directory
.	:	current directory
..	:	parent directory

- If a file path begins with **/**, then it is an ***absolute path***. It doesn't depend on the current working directory.
- If a file path begins **WITHOUT** **/**, then it is a ***relative path***. The path is defined according to the current directory. So, the path **depends** on the current working directory.

Examples

`/tmp/bootcamp/bootcamp.pl` : Absolute path.

`bootcamp.pl` : Relative path. This is the `botcamp.pl` file in the **working** directory.

`../bootcamp.pl` : Relative path. This is the `botcamp.pl` file in the **parent** directory of the working directory.

`~/bootcamp.pl` : This is the file in the **home** directory of the current user.

Files

Technically, every file is a binary file.

Yet, it is common practice to group the into two:

`Text Files`	:	Contains only printable characters.
`Binary Files`	:	Files that are not text files.

Viewing text files

Text files are of extreme importance in bioinformatics.
There are many tools to view and edit text files.
`less` is a very useful program.

```
$ less text_file
```

For details **RTFM**.

File Permissions

A typical output of

```
$ ls -lh
```

is

```
-rw-r–r–       1   ho86w   moore    11M   Nov 5    00:41   RPKM.csv
-rwxrwxrwx     1   ho86w   moore   143B   Oct 27   02:53   my_script.sh
drwxr-xr-x    21   ho86w   moore   714B   Jan 24   13:52   sam_files
```

$$\underbrace{\text{rwx}}_{\text{user}}\ \underbrace{\text{r-x}}_{\text{group}}\ \underbrace{\text{r- -}}_{\text{others}}$$

User	:	Has read, write and execute access.
Group	:	Has read and execute access but can’t write to the file
Others	:	Has only read access.

Changing File Permissions

`chmod:` Change file mode bits.

```
$ chmod filemode file
```

To give only read access to other users,

```
$ chmod o=r myscript.sh
```

To give read and execution access together,

```
$ chmod o=rx myscript.sh
```

Installing Software

What if we need a software that we dont't have in the mghpc?

You can only install software **LOCALLY!**

There may be an easier way out!

the module system

Installing Software

What if we need a software that we dont’t have in the mghpc?

You can only install software **LOCALLY!**

There may be an easier way out!

the module system

The Module System in MGHPC

Many useful bioinformatics tools are already installed!
You need to *activate* the ones you need for your account.

To see the available modules:

```
$ module avail
```

To load a module, say R version 3.0.1:

```
$ module load R/3.0.0.1
```

If you can't find the software among the available modules, you can make a request to the admins via **`ghpcc@list.umassmed.edu`**

Editing Text Files

For a very simple text editor:

```
$ nano
```

`vi` and `emacs` are very powerful text editors with sophisticated features.

Searching text files

Problem

Say, we have our RNA-Seq data in fastq format. We want to see the reads having three consecutive A's. How can we save such reads in a separate file?

grep is a program that searches the standard input or a given text file *line-by-line* for a given text or pattern.

$$\texttt{grep}\quad \underbrace{\texttt{AAA}}_{\text{text to be searched for}}\quad \underbrace{\texttt{control.rep1.1.fq}}_{\text{Our text file}}$$

For a colorful output, use the `-color` option.

```
$ grep AAA control.rep1.1.fq -color
```

What about saving the result?

We can make grep print all the reads we want on the screen.

But how can we save them? View them better?

For this we need to redirect the **standard output** to a textfile.

```
$ grep AAA control.rep1.1.fq > ~/AAA.txt
```

Unix Pipes

Problem

Instead of saving the result of grep, how can we see it in less?

Unix pipes can be used to feed the output of a program to another program.

```
$ grep AAA control.rep1.1.fq | less
```

grep: filtering out

Say we want to find reads that **don't** contain AAA in a fasta file, then we use the `-v` option to filter out reads with AAA.

```
$ grep -v AAA file.fa
```

More on Text Filtering

Problem

*How can we get **only** the nucleotide sequences in a fastq file?*

Problem

How can we get only particular columns of a file?

awk

awk is an interpreted programming language desgined to process text files. We can still use awk while staying away from the programming side.

$$\texttt{awk}\ \underbrace{\texttt{'\{print(\$2)\}'}}_{\text{awk statement}}\ \underbrace{\texttt{genes.gtf}}_{\text{text file with columns sep. by spaces}}$$

Example

Say, we only want to see the second and fourth columns in a gtf file,

```
$ awk '{print($2,$4)}'
```

Getting nucleotide sequences from fastq files

In fastq files, there are 4 lines for each read. The nucleotide sequence of the reads is on the second line respectively. We can get them using a very simple modular arithmetic operation,

```
$ awk '{if(NR % 4== 2)print($0)}' file.fq
```

NR = line number in the given file.

Unix pipes

awk can be very useful when combined with other tools.

Problem

How many reads are there in our fastq file that don't have the seqeunce ***GC****?*

```
$  awk '{if(NR % 4== 2)print($0)}' file.fq | grep -v GC
```

gives us all such reads. How do we find the number of lines in the output?

wc

wc: gives us the number of lines, words, and characters in a line.

with the `-l` olption, we only get the number of lines.

```
$   awk '{if(NR % 4== 2)print($0)}' file.fq | grep -v GC | wc -l
```

Example

Let’s find the number of chromosomes in the mm10.fa file.
Each chromosome entry begins with ">", we get them by

```
$ grep ">"
```

Then we count the number of lines

```
$ grep ">" | wc -l
```

Exercise

Let's find all files in the bootcamp directory that the group have read but not write permissions.
First, we need the first column of

```
$ ls -l
```

Then we need, the two characters starting at the fifth position. For this, we can use the substring function in awk.

`substr(text, n,m)` gives the m consecutive characters starting at the *m*th position in text. So,

```
$ ls -l | awk '{print(substr($1,5,2))}'
```

So, we are usre that we have the correct piece of the permissions. Then

```
$ ls -l | awk '{if((substr($1,5,2)=="r-")print($9)}'
```

Further Reading

Learning regular expressions is very useful for text processing. `awk, grep, vi` and many other tools can understand regular expressions.

For example, using regular expressions, one can find all entries having at least three, at most five consecutive A's in a fastq file.

Working with Compressed Files

We use

`gzip` to compress / decompress files

`tar` to pack files into one file for archiving purposes.

To compress a file `gzip file`

To decompress, `reads.fastq.gz`,

```
$ gzip -d reads.fastq.gz
```

tar

To pack a directory

```
$ tar -cvf arcive.tar directory
```

To pack a directory in a zipped tar file

```
$ tar -czvf archive.tar.gz directory
```

To get our directory back from the tar file

```
$ tar -xvf archive.tar
```

To get our directory back from the tar.gz file

```
$ tar -xvzf archive.tar.gz
```

Processes

You can list **your** running processes

```
$ ps -u username
```

where `username` is your username. You can see them in more detail with the `-F` option

```
$ ps -u username -F
```

Terminating a process

You can terminate a process by

```
$ kill -9 processID
```

where `processID` is the process id that can be obtained by the `ps` command.

Background Processes

To send a process to the background, first put the process on hold by pressing `ctrl` and z, and then send it to the background by

```
$ bg
```

You can check your processes by

```
$ jobs
```

To put a back to the foreground, first get the process id by

```
$ jobs
```

and then

```
$ fg % processNumber
```

To start a process on the background, put & at the end of your command

```
$ command &
```

Customizing Bash

Say, we want to run `hello.sh` inside any directory by simply

```
$ hello.sh
```

How can we do this? We need to add this to our path variable

```
$ PATH=/home/username/bootcamp
```

Houston! We have a problem!

Now, none of `ls` , `nano`, `...` `etc` work. What do we do?

Customizing the command prompt

The command prompt can be changed by modifying the `PS1` variable. Try

```
$ PS1="\W > "
```