An Introduction to the Command Line Interface

Charles Rahal and Felix Tropf
University of Oxford, Department of Sociology

26th June, 2017

NCRM Summer School - University of Oxford

Slides, code, lecture notes: https://github.com/crahal/Teaching

Comments/questions/suggestions: charles.rahal@sociology.ox.ac.uk

Some Class Administration

- **WELCOME!** - to the NCRM Summer School in SocioGenomics!

Some Class Administration

- **WELCOME!** - to the NCRM Summer School in SocioGenomics!

- I'd like to also introduce `Felix`: the TA for this class.

Some Class Administration

- **WELCOME!** - to the NCRM Summer School in SocioGenomics!

- I'd like to also introduce Felix: the TA for this class.

- This is the first of several computer workshops - each focusing on different things.

Some Class Administration

- **WELCOME!** - to the NCRM Summer School in SocioGenomics!

- I'd like to also introduce `Felix`: the TA for this class.

- This is the first of several computer workshops - each focusing on different things.

- Other classes are different, involving the practical use of the tools described here.

Some Class Administration

- **WELCOME!** - to the NCRM Summer School in SocioGenomics!

- I'd like to also introduce Felix: the TA for this class.

- This is the first of several computer workshops - each focusing on different things.

- Other classes are different, involving the practical use of the tools described here.

- From the pre-sessional materials, you should have set up a CLI:

  - **Windows:** this will involve a VM. If you haven't managed this, I'll help you directly after this session or this evening before or after dinner.
  - **Linux or Mac:** you already have a command line – nothing extra required.

Some Class Administration

- **WELCOME!** - to the NCRM Summer School in SocioGenomics!

- I'd like to also introduce Felix: the TA for this class.

- This is the first of several computer workshops - each focusing on different things.

- Other classes are different, involving the practical use of the tools described here.

- From the pre-sessional materials, you should have set up a CLI:

    - **Windows:** this will involve a VM. If you haven't managed this, I'll help you directly after this session or this evening before or after dinner.
    - **Linux or Mac:** you already have a command line – nothing extra required.

- However: for a lot of classes, this isn't necessary, where a simple R (more often) or Python (less often) will be more than sufficient in your Windows environment.

Some Class Administration

- **WELCOME!** - to the NCRM Summer School in SocioGenomics!

- I'd like to also introduce Felix: the TA for this class.

- This is the first of several computer workshops - each focusing on different things.

- Other classes are different, involving the practical use of the tools described here.

- From the pre-sessional materials, you should have set up a CLI:

  - **Windows:** this will involve a VM. If you haven't managed this, I'll help you directly after this session or this evening before or after dinner.
  - **Linux or Mac:** you already have a command line – nothing extra required.

- However: for a lot of classes, this isn't necessary, where a simple R (more often) or Python (less often) will be more than sufficient in your Windows environment.

- The CLI will only be required for some classes, but practicing with it this week provides **invaluable** experience for HPCs (an integral part of genomics).

Why Use the Command Line?

Lets first further motivate *why* we want to use the command line. Although there are many reasons for utilizing it, and conversely many reasons for maintaining a GUI for a lot of tasks, CLIs are important for these reasons especially:

Why Use the Command Line?

Lets first further motivate *why* we want to use the command line. Although there are many reasons for utilizing it, and conversely many reasons for maintaining a GUI for a lot of tasks, CLIs are important for these reasons especially:

- Genetics datasets are **BIG**. The just released UK Biobank dataset is about 8tb. This necessitates the use of **H**igh **P**erformance **C**omputers (HPCs). The recent flood of genetic data is due to the reduced cost of genomic sequencing (specifically NGS) from around 2008.

Why Use the Command Line?

Lets first further motivate *why* we want to use the command line. Although there are many reasons for utilizing it, and conversely many reasons for maintaining a GUI for a lot of tasks, CLIs are important for these reasons especially:

- Genetics datasets are **BIG**. The just released UK Biobank dataset is about 8tb. This necessitates the use of **H**igh **P**erformance **C**omputers (HPCs). The recent flood of genetic data is due to the reduced cost of genomic sequencing (specifically NGS) from around 2008.

- Automation/Replication: No matter how complex the command or series of commands, the CLI doubles as a scripting language (shell scripts - which we will briefly discuss if there's time) which can be ran as a batch.

Why Use the Command Line?

Lets first further motivate *why* we want to use the command line. Although there are many reasons for utilizing it, and conversely many reasons for maintaining a GUI for a lot of tasks, CLIs are important for these reasons especially:

- Genetics datasets are **BIG**. The just released UK Biobank dataset is about 8tb. This necessitates the use of **H**igh **P**erformance **C**omputers (HPCs). The recent flood of genetic data is due to the reduced cost of genomic sequencing (specifically NGS) from around 2008.

- Automation/Replication: No matter how complex the command or series of commands, the CLI doubles as a scripting language (shell scripts - which we will briefly discuss if there's time) which can be ran as a batch.

- Control: Commands are often more powerful and precise, giving more control over the operations to be performed

Why Use the Command Line?

Lets first further motivate *why* we want to use the command line. Although there are many reasons for utilizing it, and conversely many reasons for maintaining a GUI for a lot of tasks, CLIs are important for these reasons especially:

- Genetics datasets are **BIG**. The just released UK Biobank dataset is about 8tb. This necessitates the use of **H**igh **P**erformance **C**omputers (HPCs). The recent flood of genetic data is due to the reduced cost of genomic sequencing (specifically NGS) from around 2008.

- Automation/Replication: No matter how complex the command or series of commands, the CLI doubles as a scripting language (shell scripts - which we will briefly discuss if there's time) which can be ran as a batch.

- Control: Commands are often more powerful and precise, giving more control over the operations to be performed

- Software tools: A number of genetics software tools will only run in Linux/Mac environments.

Our Friend - ARCUS (or AWS)

- Why have this class at all? Our friend ARCUS provides some motivation:

  > **WARNING**
  >
  > Finally, Linux assumes that the user knows what they are doing. On ARC systems users will not have escalated or 'root' privileges but will be able to delete or modify any files/data that resides in their own home account or project data directory. Where you are sharing data with other project members you need to be careful when using Linux commands as you may inadvertently delete or modify this data. Most Linux (UNIX) commands do not ask questions. They assume you know what you are typing and blindly execute the task as instructed.

Our Friend - ARCUS (or AWS)

- Why have this class at all? Our friend ARCUS provides some motivation:

> **WARNING**
>
> Finally, Linux assumes that the user knows what they are doing. On ARC systems users will not have escalated or 'root' privileges but will be able to delete or modify any files/data that resides in their own home account or project data directory. Where you are sharing data with other project members you need to be careful when using Linux commands as you may inadvertently delete or modify this data. Most Linux (UNIX) commands do not ask questions. They assume you know what you are typing and blindly execute the task as instructed.

- The CLI can seem like a harsh, unforgiving environment.

Our Friend - ARCUS (or AWS)

- Why have this class at all? Our friend ARCUS provides some motivation:

  > **WARNING**
  >
  > Finally, Linux assumes that the user knows what they are doing. On ARC systems users will not have escalated or 'root' privileges but will be able to delete or modify any files/data that resides in their own home account or project data directory. Where you are sharing data with other project members you need to be careful when using Linux commands as you may inadvertently delete or modify this data. Most Linux (UNIX) commands do not ask questions. They assume you know what you are typing and blindly execute the task as instructed.

- The CLI can seem like a harsh, unforgiving environment.

- Urban Dictionary calls the famous CLI command `rm -rf`:

  > *'the finest compression available under UNIX/Linux...*
  > *unfortunately, there is no decompresser available.'*

Our Friend - ARCUS (or AWS)

- Why have this class at all? Our friend ARCUS provides some motivation:

> **WARNING**
>
> Finally, Linux assumes that the user knows what they are doing. On ARC systems users will not have escalated or 'root' privileges but will be able to delete or modify any files/data that resides in their own home account or project data directory. Where you are sharing data with other project members you need to be careful when using Linux commands as you may inadvertently delete or modify this data. Most Linux (UNIX) commands do not ask questions. They assume you know what you are typing and blindly execute the task as instructed.

- The CLI can seem like a harsh, unforgiving environment.

- Urban Dictionary calls the famous CLI command `rm -rf`:

> *'the finest compression available under UNIX/Linux...*
> *unfortunately, there is no decompresser available.'*

- However: **the potential is endless!**

Our Friend - ARCUS (or AWS)

- Why have this class at all? Our friend ARCUS provides some motivation:

> **WARNING**
>
> Finally, Linux assumes that the user knows what they are doing. On ARC systems users will not have escalated or 'root' privileges but will be able to delete or modify any files/data that resides in their own home account or project data directory. Where you are sharing data with other project members you need to be careful when using Linux commands as you may inadvertently delete or modify this data. Most Linux (UNIX) commands do not ask questions. They assume you know what you are typing and blindly execute the task as instructed.

- The CLI can seem like a harsh, unforgiving environment.

- Urban Dictionary calls the famous CLI command `rm -rf`:

> *'the finest compression available under UNIX/Linux...*
> *unfortunately, there is no decompresser available.'*

- However: **the potential is endless!**

- (and a Virtual Machine is the best way to learn).

This isn't a scary course!

- Melinda told me that a good strategy is to put a picture of your cat on the slides to persuade participants that the course isn't scary...

This isn't a scary course!

- Melinda told me that a good strategy is to put a picture of your cat on the slides to persuade participants that the course isn't scary...

Figure: This isn't a scary course!

(a) Sleepy

(b) Curious

A Brief History of nix*-like environments

- Before we discuss *exactly* what the CLI is, we should briefly talk about the history of what is now known as *nix-like computing.

A Brief History of nix*-like environments

- Before we discuss *exactly* what the CLI is, we should briefly talk about the history of what is now known as *nix-like computing.

- Unix was created 1969 at Bell Laboratories – then a development division at AT&T – by Ken Thompson and Dennis Richie.

A Brief History of nix*-like environments

- Before we discuss *exactly* what the CLI is, we should briefly talk about the history of what is now known as *nix-like computing.

- Unix was created 1969 at Bell Laboratories – then a development division at AT&T – by Ken Thompson and Dennis Richie.

- All software previously hardware dependent, written in assembler language.

A Brief History of nix*-like environments

- Before we discuss *exactly* what the CLI is, we should briefly talk about the history of what is now known as *nix-like computing.

- Unix was created 1969 at Bell Laboratories – then a development division at AT&T – by Ken Thompson and Dennis Richie.

- All software previously hardware dependent, written in assembler language.

- However – a critical feature of Unix was that it is proprietary.

A Brief History of nix*-like environments

- Before we discuss *exactly* what the CLI is, we should briefly talk about the history of what is now known as *nix-like computing.

- Unix was created 1969 at Bell Laboratories – then a development division at AT&T – by Ken Thompson and Dennis Richie.

- All software previously hardware dependent, written in assembler language.

- However – a critical feature of Unix was that it is proprietary.

- University of California, Berkley developed BSD – based on the AT&T code base. Commercial Unix derivatives emerged through the 1980s (i.e. Mac OS).

A Brief History of nix*-like environments

- Before we discuss *exactly* what the CLI is, we should briefly talk about the history of what is now known as *nix-like computing.

- Unix was created 1969 at Bell Laboratories – then a development division at AT&T – by Ken Thompson and Dennis Richie.

- All software previously hardware dependent, written in assembler language.

- However – a critical feature of Unix was that it is proprietary.

- University of California, Berkley developed BSD – based on the AT&T code base. Commercial Unix derivatives emerged through the 1980s (i.e. Mac OS).

- The breakthrough came in September 1991, when Linus Torvalds released the Linux operating system kernel.

A Brief History of nix*-like environments

- Before we discuss *exactly* what the CLI is, we should briefly talk about the history of what is now known as *nix-like computing.

- Unix was created 1969 at Bell Laboratories – then a development division at AT&T – by Ken Thompson and Dennis Richie.

- All software previously hardware dependent, written in assembler language.

- However – a critical feature of Unix was that it is proprietary.

- University of California, Berkley developed BSD – based on the AT&T code base. Commercial Unix derivatives emerged through the 1980s (i.e. Mac OS).

- The breakthrough came in September 1991, when Linus Torvalds released the Linux operating system kernel.

- **Utilized everywhere:** i.e. Android operating system and Chrome OS which runs on Chromebooks (televisions, smart-watches, servers, etc).

The File Tree
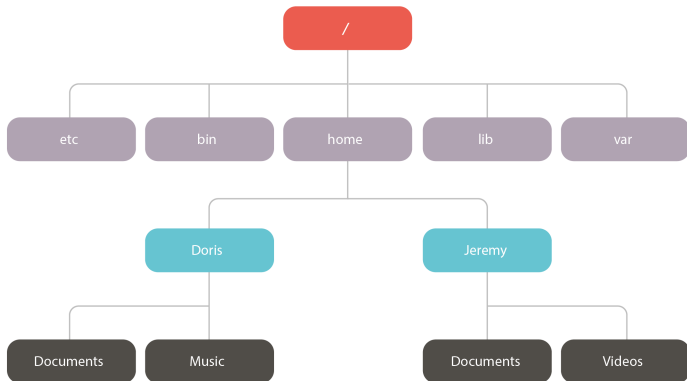
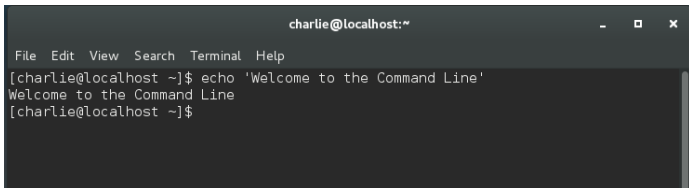- You might also find /opt: optional software or /tmp: temporary space.

System Architecture
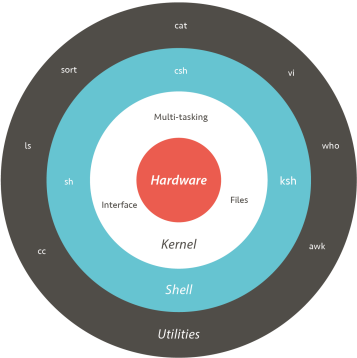
- The shell is a friendly interface that translates your commands into some low-level calls to the kernel.

The Terminal

- The terminal is a program which lets you interact with the shell:

- If the last character of your shell prompt is # rather than $, you are operating as the superuser. This means that you have administrative privileges. This can be potentially dangerous, since you are able to delete or overwrite any file on the system. Unless you absolutely need administrative privileges, do not operate as the superuser (try whoami and sudo whoami).

Fundamental CLI Commands: `man`

- The `man` command shows the manual for a given page, including information on options and usage.

- This is the default resource for getting help on specific commands.

The `man` Command

```
1  user@system:~$ man echo
2  ECHO(1) User Commands ECHO(1)
3
4  NAME
5         echo − display a line of text
6
7  SYNOPSIS
8  ...
```

- For example, the `man echo` command will display the manual for the `echo` command which we just saw.

Fundamental CLI Commands: `whatis`

- The `whatis` command gives a short summary description of the specific command, and command inputs can be stacked together.

The `whatis` Command

```
1 user@system:~$ whatis echo ls
2 echo (1) − display a line of text
3 ls (1) − list directory contents
```

- Each manual page has a short description available within it – and `whatis` searches the manual page names.

Fundamental CLI Commands: `ls`

- `ls` is a Linux shell command that lists directory contents of files and directories.

The `ls` Command

```
1 user@system:~$ ls
2 Desktop Documents Downloads Music Pictures Public Videos
```

- The lists can also be sorted, accept wildcards, and pipe outputs to file.

- It can show hidden files, show file size, and has the option for a 'long' format.

Fundamental CLI Commands: pwd

- However, this isn't much use without knowing *where* we are in the file tree.

The pwd Command

```
1 user@system:~$ pwd
2 /home/user
```

- In *nix-like operating systems, the pwd command (which stands for print working directory) writes the full path-name of the current working directory to the standard output.

Fundamental CLI Commands: `mkdir` and `cd`

- We should make a new directory for the summer school.

  The `mkdir` and `cd` Commands

  ```
  1 user@system:~$ mkdir NCRM
  2 user@system:~$ ls
  3 NCRMTextbookChapter
  4 user@system:~$ cd NCRM
  5 user@system:~/NCRM$ pwd
  6 /home/user/NCRM
  ```

- This brings us to the concept of *absolute* and *relative* paths.

- From the documents folder above, we can navigate to the new folder (NCRM) through the relative path or through the absolute path.

- The cd command is used to change the current directory.

- There are multiple ways of identifying the same file path in the directory tree.

Fundamental CLI Commands: `touch`

- The `touch` command is the easiest way to create new, empty files...

The `touch` Command

```
1 user@system:~/NCRM$ touch testfile
2 user@system:~/NCRM$ ls
3 testfile
```

- `testfile` might seem unfamiliar to you for one obvious reason: the lack of an extension.

- In *nix-like systems, the extension is ignored, and the file type is determined automatically (and the command `file` can tell us additional information).

Fundamental CLI Commands: `rm`

- Now that we can create files and directories, lets look at removing them:

The `rm` Command

```
1 user@system:~/NCRM$ cd ..
2 user@system:~/$ rm −ri NCRM
3 rm: remove directory 'NCRM'?
```

- This shows how we can stack options together, where `rm -ri` is the equivalent to `rm -r -i`.

- However, command line options are not universal between commands, and implementation across different operating systems may vary.

- We should also note the existence of `rmdir`, the negative equivalent to `mkdir`, which removes *empty* directories.

- Note the use of `..` to move up a directory (and many other similar shortcuts).

Fundamental CLI Commands: `mv` and `cp`

- It seems only natural that we now learn how to move, copy and rename files.

The `mv` and `cp` Commands

```
1 user@system:~/NCRM$ cp testfile ..
2 user@system:~/NCRM$ rm ../testfile
3 user@system:~/NCRM$ mv testfile ..
```

- We need not actually remove `testfile` after copying it and before moving it, as the `mv` command would simply overwrite it.

- To rename a file, we can just move it with a new name.

Other Basic Utilities

- We can clear the terminal screen.

- We can display all environmental variables with env (try echo $LANG).

- find and locate search for files and directories (with the latter being faster – performing on a database of indexed filenames).

- date displays or sets (with the option -s) the system time, and cal displays the calender.

- history and specifically - history [NUM] reports the last [NUM] commands.

- We can exit from the current terminal session, logout as a specific user, or shutdown the machine entirely.

- We can also tar items together, and then gzip or bzip2 them up.

Editors at the Command Line

- Lets introduce the use of text editors at the command line (vi, pico, nano and emacs). These are plain text editors – not word processing suites.

- Unlike many GUI based editors, the mouse does not move the cursor. Unlike PC editors, you cannot modify text by highlighting it with a mouse.

- Lets use `nano` to create a file which we will use for some examples later on: a list of all the fruits we can think of (call the file `allfruits`).

Introducing Text Editors at the CLI: `nano allfruits`

```
1  Apple
2  Apricot
3  Avocado
4  .
5  .
6  "allfruits" 90 lines, 846 characters
```

I/O Redirection

- Most command line programs output their results to the 'standard output' (STDOUT): which, by default, is the display.
- We can redirect standard output to specific files using the '$>$' character, and append to a file using '$>>$'.
- Commands can also accept input from 'standard input' (STDIN), which defaults to the keyboard: to redirect from STDIN, we use the '$<$' character.

STDOUT and STDIN

```
1 user@system:~/NCRM$ ls > filelist.txt
2 user@system:~/NCRM$ sort < filelist.txt
```

- There is also a third stream which will go unexamined ('standard error' or STDERR) which is used for error messages – and also defaults to the terminal.

Filters

filters are a class of programs which are extremely useful for I/O Redirection:

- `sort`: Sorts STDIN and outputs the sorted result on standard output.

- `uniq`: Given a sorted stream of data from STDIN, it removes duplicates.

- `head`: Outputs the first few lines of its input (defaults to first 10 lines).

- `cat`: Concatenates files and displays their contents, or just displays contents if given one input.

- `less`: used to view (but not change) the contents of a text file one screen at a time.

- `cut`: Divides a file into several columns.

- `nl`: prints the line number before data.

- `wc`: which prints the count of lines, words and characters.

The Pipe

- The pipe (|), which allows you to connect multiple commands together.

- The standard output of one command is fed into the standard input of another, enabling you to chain together individual commands to create something really powerful.

- The example below *pipes* the output from ls into the head command which takes the input 1 to show us the first 1 line of the ls output: we've chained the two commands together.

The Pipe (|)

```
1 user@system:~/NCRM$ ls | head −1
2 allfruits
```

Wildcards

- Another more advanced concept is the 'wildcard': a set of tools that allow you to create a pattern which defines a specific set of files or directories.

- There are two simple types of wildcards:

    1. * represents zero or more characters
    2. ? represents a single character.

Wildcards: * and ?

```
1 user@system:~/NCRM$ touch fileonea fileoneb filetwoa filetwob
2 user@system:~/NCRM$ ls file*a
3 fileonea filetwoa
4 user@system:~/NCRM$ ls filetwo?
5 filetwoa filetwob
```

- A regular expression (or 'regex') includes such functionality, but is a much more powerful pattern matcher beyond the scope of this introduction.

Grep

- One particularly useful filter is grep, which examines each line of data it receives from standard input and outputs every line that contains a specified pattern of characters.

- It utility filters input, looking for matches.

- Lets use it to find all examples of berries within our fruits file.

- In our example, we pass it the -E option, which allows it to interpret the wildcard:

The grep Command

```
1 user@system:~/NCRM$ grep −E *berry allfruits
2 strawberry
3 raspberry
4 blackberry
```

Users, Groups and Permissions

- Permissions specify what a user can and cannot do: i.e. lock your files so other people cannot change them or secure system files from damage.

- Permissions are split into three distinct categories which govern the ability to: Read: `r`, Write: `w` and Execute: `x`.

- For every file, we need to define permissions for potential users:
  - The user who created the file (u).
  - The group which owns the file (g).
  - Others (o).

- There are typically only two people who can manage the permissions of a given file or directory: the owner and the root user.

- To view the permissions associated with an individual file, we can use the `-l` option on the `ls` command:

```
1 user@system:~/NCRM$ ls −l allfruits
2 −rw−rw−r−− 1 user user 846 May 29 14:57 allfruits
```

Users, Groups and Permissions (Cont.)

- The first character determines whether it is a file (-) or a directory (d).

- We have information on the permissions for u, g and o.

- A - represents the omission of a permission.

- Change permissions with chmod, specifying: 1.) Who? 2.) Giving (+) or taking (-)?, 3.) Which permissions

- For example, lets take away read permissions from the group and others:

The chmod Command

```
1 user@system:~/NCRM$ chmod og−r allfruits
2 user@system:~/NCRM$ ls −l allfruits
3 −rw−−−−−−− 1 user user 846 May 29 15:09 allfruits
```

- Permissions are not inherited from the parent directory (unless -R: recursive).

- There are a range of 'short-hand' commands (e.g. chmod 751 <filename>).

Bash Scripting: Introduction

- Bash scripting performs complex, repetitive tasks with minimal effort.

- A script is just a text file containing commands which could be ran directly.

- It is a convention (albeit unnecessary) to give bash scripts an extension of .sh.

Our first script: `myfirstscript.sh`

```
1  #!/bin/bash
2  echo Hello There! Welcome to Bash Scripting!
```

- The first line is called the 'shebang'. We can run the file in two ways:

Executing `myfirstscript.sh`

```
1  user@system:~/NCRM$ ./myfirstscript.sh
2  Hello There! Welcome to Bash Scripting!
3  user@system:~/NCRM$ bash myfirstscript.sh
4  Hello There! Welcome to Bash Scripting!
```

Bash Scripting: Variables

- Just like in other languages: variables are temporary methods of storing information.

- Setting them requires no $, but reading them does.

- Lets pass variables to a script which accepts $1 and $2 as inputs:

Accepting Variables: `mysecondscript.sh`

```
1 #!/bin/bash
2 echo Hello $1! Message sent from $2!
```

and then execute it as before:

Executing `mysecondscript.sh`

```
1 user@system:~/NCRM$ ./variables.sh Felix Charlie
2 Hello Felix! Message sent from Charlie!
```

- There are also a number of special variables: $0, $n, $? etc.

Bash Scripting: User Input

- We can also ask the user for input using read:

Asking for User Input: mythirdscript.sh

```
1  #!/bin/bash
2  echo Hey, Buddy! What is your favorite color?
3  read favoritecolor
4  echo Wow! $favoritecolor is my favorite color too!
```

and then execute it as before:

Executing mythirdscript.sh

```
1  user@system:~/NCRM$ ./readvariable.sh
2  Hey, Buddy! What is your favorite color?
3  Blue
4  Wow! Blue is my favorite color too!
```

Bash Scripting: Arithmetic

- Do simple arithmetic operations with let (store) and expr (print). We can utilize the standard operators of +, -, *, /.

Simple Arithmetic Operations: myfourthscript.sh

```
1 #!/bin/bash
2 let ''a = $1 + $2''
3 echo adding $1 and $2 together gives $a
4 echo multiplying $1 and $2 together gives $(expr $1 \* $2)
```

and then execute it as before:

Executing myfourthscript.sh

```
1 user@system:~/NCRM$ ./arithmetic.sh 10 5
2 adding 10 and 5 together gives 15
3 multiplying 10 and 5 together gives 50
```

- Finally, we can also embed if statements just like in other languages (no indentation), and utilize various types of loops (while, until and for).

Python and R

- HPCs requires us to submit our Python and R scripts through the CLI.

- This is as simple as writing your R or Python scripts locally and then transferring them (scp or rsync) or writing them using a CLI editor.

- Lets create an example file called helloworld which simply has the single line: print(''Hello World! How are you?'').

- Assuming Py 2, we can execute helloworld with almost identical output:

Executing Python and R Scripts in the CLI

```
1 user@system:~/NCRM$ python helloworld
2 Hello World! How are you?
3 user@system:~/NCRM$ Rscript helloworld
4 [1] ''Hello World! How are you?''
```

- We can execute the files with ./ as above, but this requires a shebang!
  #!/usr/bin/Rscript for R and #!/usr/bin/env python.

Conclusion

- We've gone through the main basic ideas behind the CLI, with the intention of providing some intuition.

Conclusion

- We've gone through the main basic ideas behind the CLI, with the intention of providing some intuition.

- Other courses use these ideas in their application to sociogenomics.

Conclusion

- We've gone through the main basic ideas behind the CLI, with the intention of providing some intuition.

- Other courses use these ideas in their application to sociogenomics.

- We didn't really cover: alias-ing, networking, **ssh-ing**, scheduling, `awk` or `sed`, and we didn't fully consider control statements, or regex.

Conclusion

- We've gone through the main basic ideas behind the CLI, with the intention of providing some intuition.

- Other courses use these ideas in their application to sociogenomics.

- We didn't really cover: alias-ing, networking, **ssh-ing**, scheduling, awk or sed, and we didn't fully consider control statements, or regex.

- Hopefully you now feel as suave with the CLI as this guy: