Introduction to Reproducible Research and CLI

James Balamuta

Department of Informatics, Statistics
University of Illinois at Urbana-Champaign

On the Agenda

1. Reproducible Research
   - Definition
   - Being a Practitioner
   - Workflow
2. Tools of Reproducible Research
   - Overview
   - Editors
3. CLI
   - Intro Shell
   - File System
   - Using Commands
   - Directory Commands
   - File Commands
   - File Permissions

Ready?

On the Agenda

What is Reproducible Research?

Reproducible research or creating a **reproducible analysis** is the idea that the experiment's collected data, data analysis code, and derived principal results are assembled in a way so that another body is able to re-create all of the results (e.g., data formatting, parameter estimates, figures, tables, and so on).

In essence, reproducible research seeks to satisfy a very minimal portion of how to obtain *replicable* results typically used to promote scientific theories.

Reproducible vs. Replicable

In general, there are lots of papers that debate what the definitions of Reproducible and Replicable are.

For our purpose, we will consider the viewpoint of Prof. Roger Peng of the Journal of Biostatistics - held as the Journal's standard - and echoed by Prof. David Banks, former editor of the prestigious Journal of the American Statistical Association (JASA).

Reproducible if there is a specific set of computational functions/analyses (usually specified in terms of code) that exactly reproduce all of the numbers in a published paper from raw data.

Replicable if you perform the exact same experiment (at least) twice, collect data in the same way both times, perform the same data analysis, and arrive at the same conclusions.

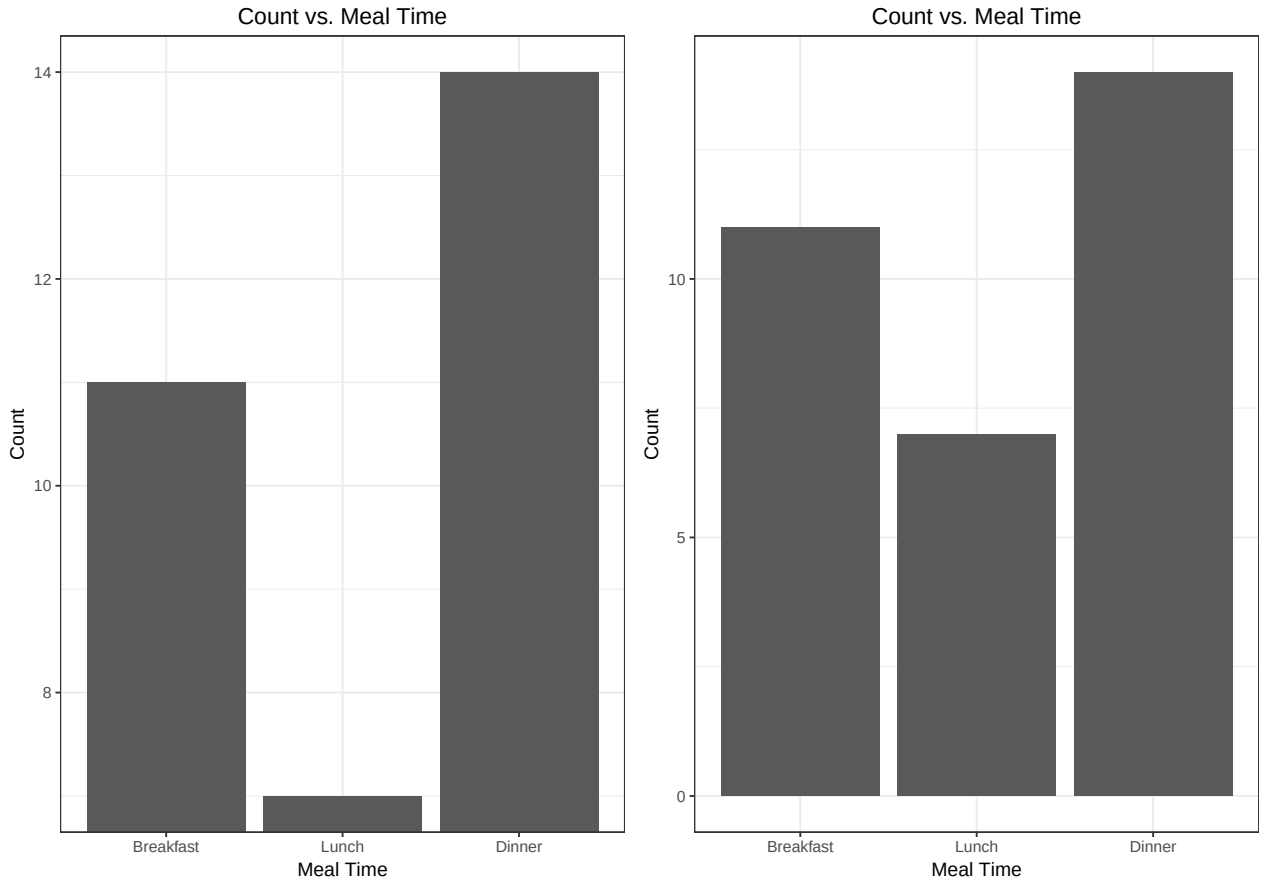
Importance of Reproducible Research

Many issues have arisen over the recent years regarding reproducibility...

- Nature: 1,500 scientists lift the lid on reproducibility:
  - "More than 70% of researchers have tried and failed to reproduce another scientist's experiments, and more than half have failed to reproduce their own experiments."
- JAMA: Contradicted and Initially Stronger Effects in Highly Cited Clinical Research
  - "Of 49 highly cited original clinical research studies, 45 claimed that the intervention was effective. Of these, 7 (16%) were contradicted by subsequent studies, 7 others (16%) had found effects that were stronger than those of subsequent studies, 20 (44%) were replicated, and 11 (24%) remained largely unchallenged.
- Nature: Over half of psychology studies fail reproducibility test
  - "Whereas 97% of the original studies found a significant effect, only 36% of replication studies found significant results."
- RetractionWatch: Tracking retractions

Lies, Damned Lies and Statistics

In the book, *How to Lie with Statistics* by Darrell Huff, a notable issue that is emphasized is the ease with which an incorrect intepretation can easily lead to an inappropriate conclusion that is **published**.



Real Life Example: Excel Breaking an Analysis

Figure 1: Austerity's Spreadsheet Error - Caught by Thomas Herndon

Why Practice Reproducible Research?

By structuring research or an analysis so that it is reproducible, not only is the work more useful but also the overload on the practitioner is reduced.

The overload is reduced since the hope of reproducible research is to put an end to the practice of copying and pasting results into documents, asymmetric data modifications in excel, and undocumented code.

As they say...

If you do something *by hand* once, you'll end up doing it at least 20 times.

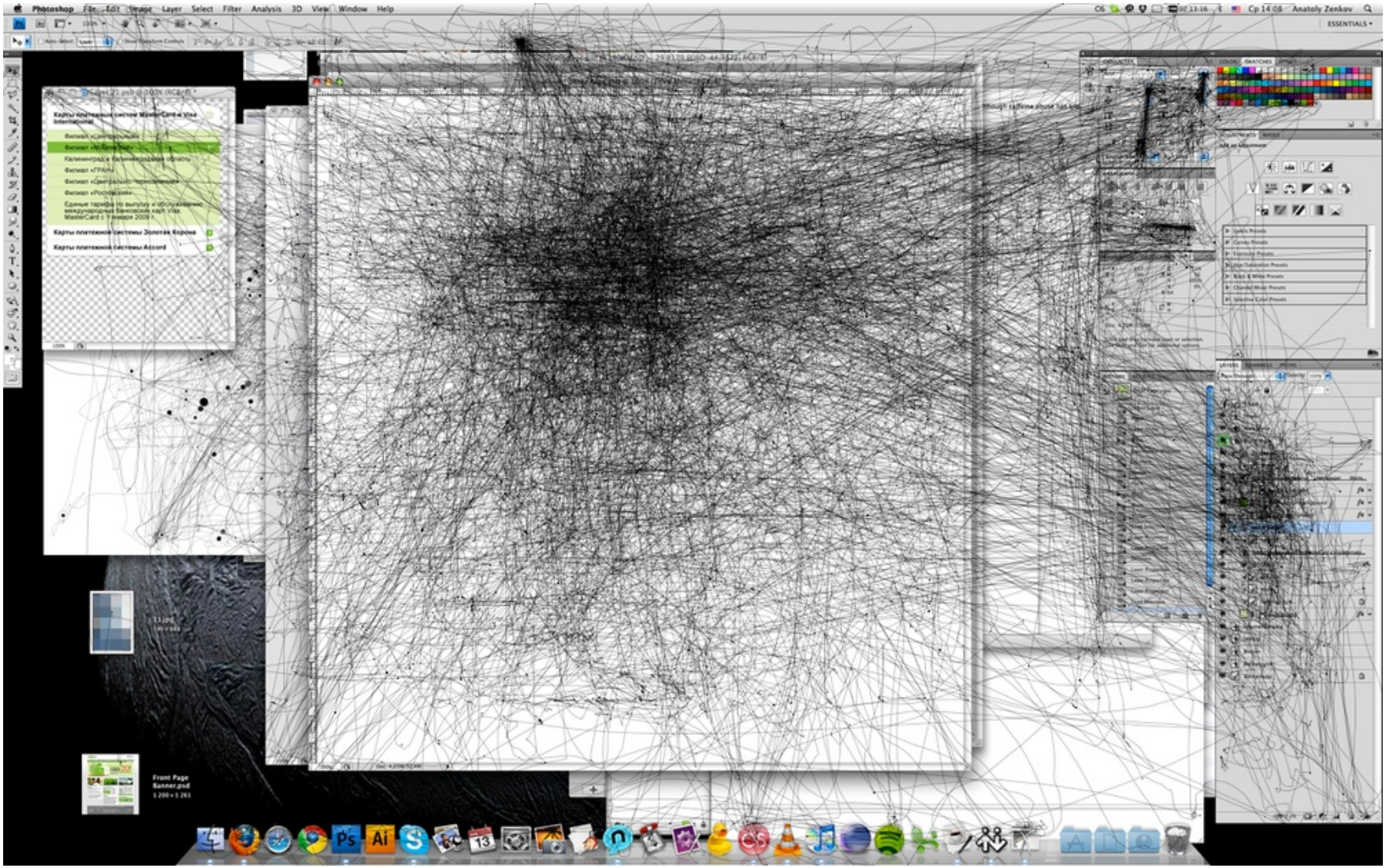
Types of Workflow

There are two ways to interact with a computer:

- **Command Line Interface (CLI):** Text-based commands issued by a keyboard that receive text-based responses from the computer.
- **Graphical User Interface (GUI):** Point-and-click command that elicits a visual response which changes the program's state.

What interface do *you* think is the preferred way to structure a *reproducible* project?

Sample Point and Click Map with Overlay

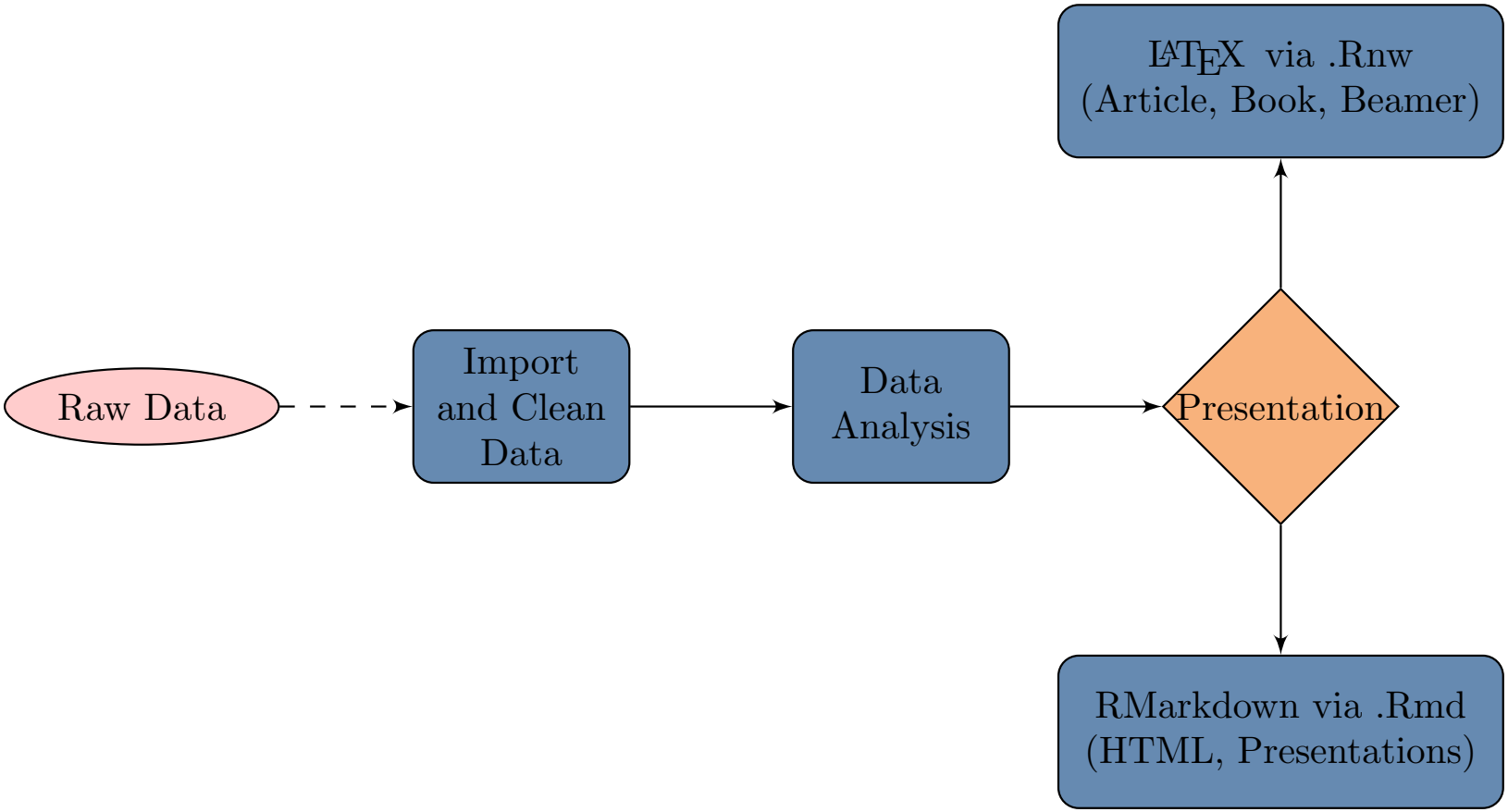


Source IOGraphica: 4 hours of Mouse Movements in Photoshop

Sample CLI History

```
Last login: Tues Jun 13 13:31:45 on ttys003
wirelesssprv-10-193-53-59:~ agentxyz$ history
    1  pwd
    2  ssh balamut2@cc-login.campuscluster.illinois.edu
    3  cp -R ~/stat385/su16/lec17 ~/stat385/su17/lec2/
    4  ls
    5  mkdir hellostat385
    6  cd hellostat385/
    7  git init
```

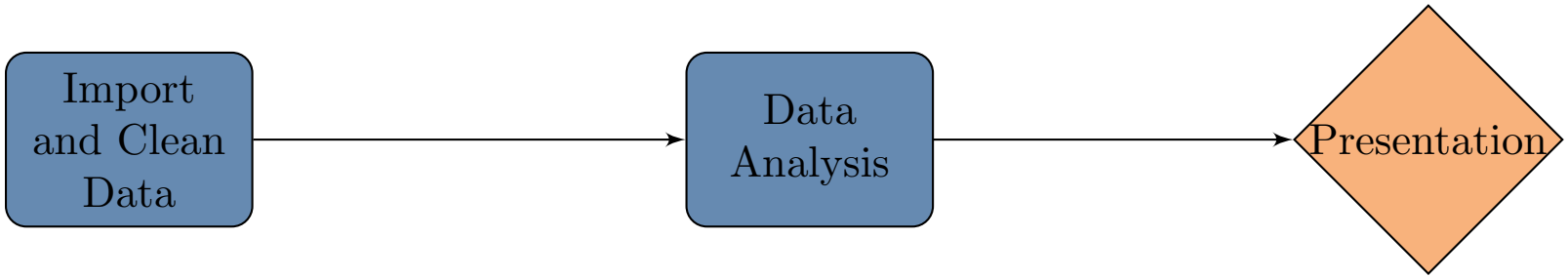
Ideal Work Flow



Only raw data exists outside of the ecosystem.

All blue boxes are done with a script to ensure reproducibility.

The Power of Scripts



- Modifications are documented
- Uniformly applied cleaning methods
- Resiliency to wrong data version

- Perform analysis like normal, but...
- No need to export figures or tables
- Code is **reusable** between projects

- Figures and tables are already created!
- Analysis changed? **Auto-updates!**
- Results are shareable and customizable

The Best Reason to Practice Reproducible Research...

James,
Hope all is well. Prof. *Toad* accidently sent us the wrong data set. Please see the forwarded e-mail and redo the analysis using the new data set. If possible, could we discuss the results on Wednesday?
Thanks,
Steven

On the Agenda

Software of Reproducible Research

The following software programs are key to Reproducible Research:

- Command Line Interfaces (CLI)
  - Windows: Cygwin, Bash on Ubuntu on Windows (Requires Win10), or use the shell that is included with Git Bash
  - macOS: Built in Terminal or iTerm2
  - Linux: Built in Shell, Terminator, or Tilda
- Version Control System (VCS): git
- Programming Language: R, Python, etc.
- Swiss Army Knife of Document Conversion: pandoc

Optional:

- LaTeX: Your Favoriate Distribution of LaTeXif you wish to compile PDFs

Code Editors

The previously listed suite of software programs left out *editors*.

Editors are applications that can be used to view, write, or modify code.

Nowadays, editors are far more power with features such as:

- Code syntax highlighting for functions and variable types;
- Fuzzy Autocompletion of function names and file paths;
- Context-specific autocompletion of variable names;
- Code diagnostics indicating potential line errors, styling issues;
- Source code search across multiple files within a project;
- Automatic line indentation and closure of parentheses;
- Reformat code to adhere to a style guide;
- Code templates for commonly used snippets of code.

Possible Code Editors

Here are a *few* programs to consider as potential candidates to write code in:

- Notepad/TextEdit: Featureless text editor
- vi/vim/nano: CLI text editors
- Notepad++/Sublime Text: GUI editors that are cross-platform
- RStudio: Integrated Developer Environment for R
  - Not great for languages *outside* of R.
- TextMate (macOS): Highly versatile GUI text editor

For the most part, examples within this course will done using either `RStudio` or the `vi` text editor.

On the Agenda

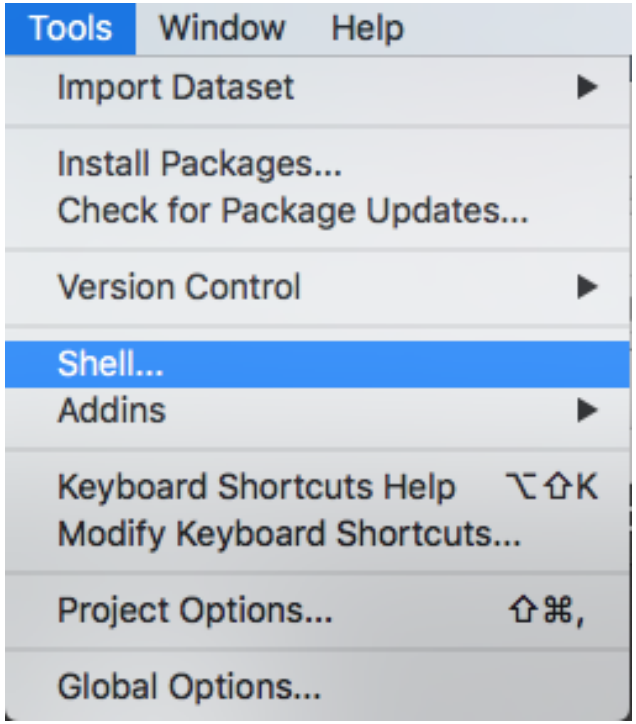
1. Reproducible Research
   - Definition
   - Being a Practitioner
   - Workflow
2. Tools of Reproducible Research
   - Overview
   - Editors
3. CLI
   - Intro Shell
   - File System
   - Using Commands
   - Directory Commands
   - File Commands
   - File Permissions

GNU Bash (Unix Shell)

- `GNU` is a free software environment that stands for "GNU's Not Unix"
  - Recursive acronym
  - Logo is a gnu head (see above)
- `Bourne-Again` Shell (`bash`) written by **Brian Fox** for the GNU Project
  - Used on most Linux operating systems and on macOS.
  - Released in 1989

Accessing `bash` in *RStudio*

- By default, this is included in RStudio:
  - `Tools` ⇒ `Shell...`



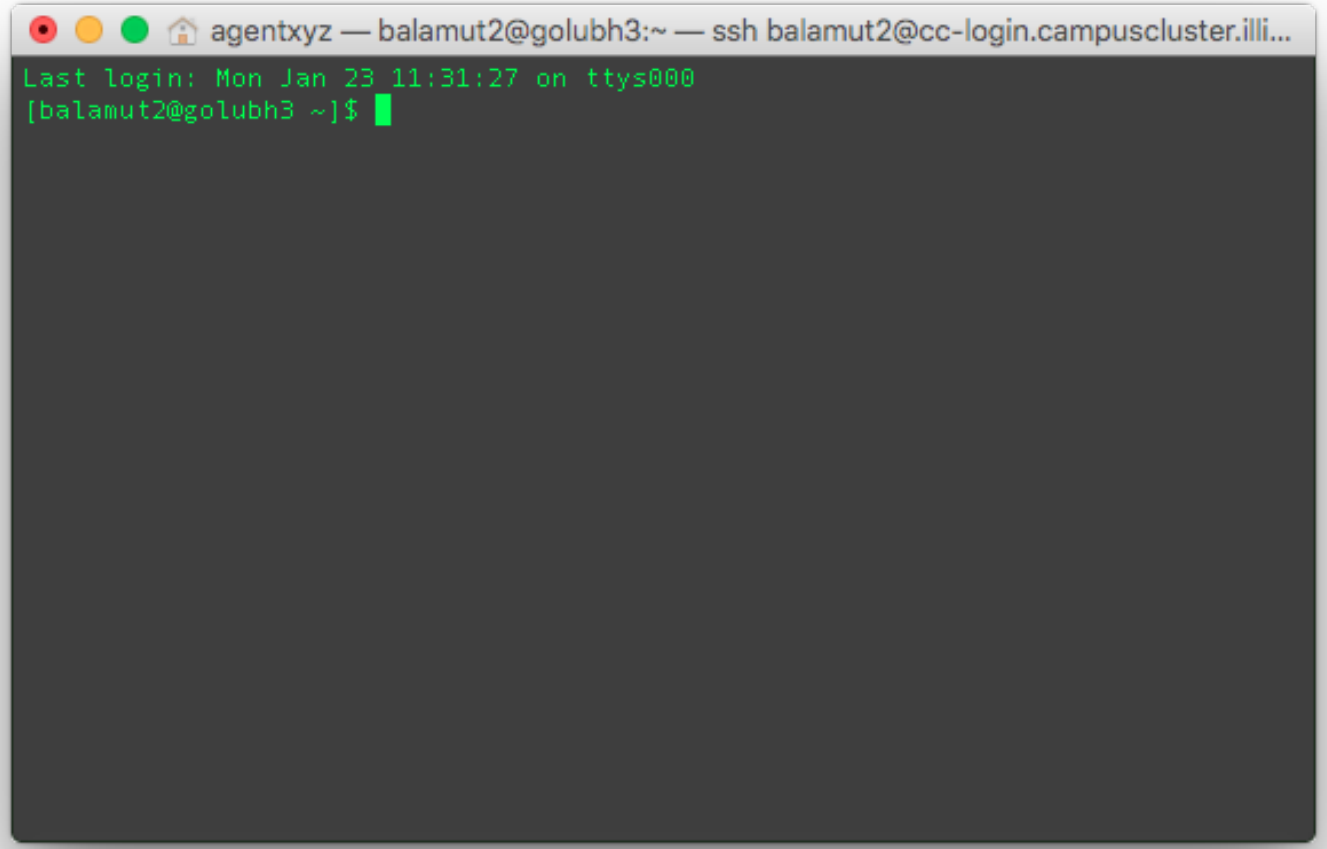
- **Note:** This may not be the case on Windows as you may only receive the command line prompt.

Grab a copy of `bash`

- If on Windows, please install the **`git bash`** shell.
  - This outside of the `GitHub Desktop`.
- In addition, Windows users may want to emulate shell in more depth with:
  - `cygwin` (instructions skip over the AWS parts); or
  - Bash on Windows / Windows Linux Subsystems (Windows 10 Required) (instructions)

Bash Shell

The bash shell looks like so:



Bash Prompt

When logged into `bash` it is traditional to see on the left hand side:

```
[username@hostname:directory]$
```

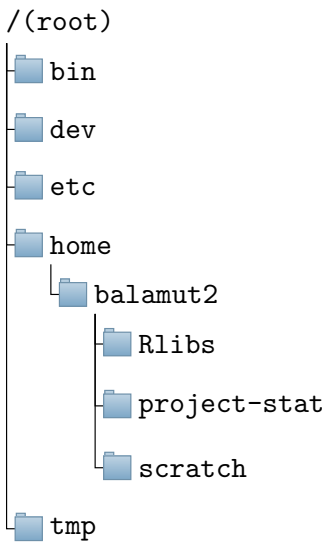
In my case, I have:

```
[balamut2@golubh3:~]$
```

- The ~ means "home directory" or `/home/username`

Directory Structure

Traditional directory structure on *Linux* is given as:



Each operating system has a different location for the user *home* directory

- Windows: `C:/Users/balamut2`
  - Cygwin: `/cygdrive/c/Users/balamut2/` or WSL: `/mnt/c/Users/balamut2/`
- macOS: `/Users/balamut2`

Path Structure

There are two forms of path structures you may encounter:

- **Fixed** or **Absolute**: Specifying the path from the root (/) directory

  `C:/Users/James/Hypno/Toad.R`
  `/etc/Renviron.site`
  `/Users/James/URA/reports/summary.docx`

- **Relative**: Resolve from the present working directory location

  `Hypno/Toad.R`
  `Renviron.site`
  `URA/reports/summary.docx`

The *best* path structure to use is **Relative**.

1. Why is relative the best?
2. What kind of path is `~/Documents/orange.txt`?

Special File System Directories

There are two special file system directory names available in **every** directory and program.

- . (period): The present directory.
- .. (two periods): The parent directory.

These special symbols provide great flexibility for *relative* paths.

Unix Prompt commands

- **Syntax:** `command [option] [source file(s)] [target file]`
  - Options often have the `-x` or `--xxx` format
  - Use `Tab` to autocomplete source file / target file name.

Advanced Unix Command Usage

Basics of the Unix Philosophy

> *Rule of Modularity: Write simple parts connected by clean interfaces.*
> — *Eric Steven Raymond (The Art of Unix Programming)*

Each command shown next is meant to address a *specific* need.

Needs are brought together by a series of operators:

- Chain operations together via pipe operator |
- Execute the next command *if* the previous one succeeds using &&
- Redirection operators <, >, >>, 2> for input/output/error

First Unix Command

Ask the computer, "Who am I?"

```
whoami
```

```
## agentxyz
```

Provides the name of the user presently logged into the shell.

Useful Unix Commands - Directories

Command	Description	Example
`pwd`	Print working directory	`pwd`
`cd`	Change directory	`cd dir/new` **or** `cd ../`
`ls`	List files	`ls ~/` **or** `ls -la new/`
`mkdir`	Make directory	`mkdir test` **or** `mkdir -p mr/r`
`rmdir`	Remove directory	`rmdir test` **or** `rmdir -p mr/r`

Unix Commands - pwd - Print working directory

```
pwd
```

```
## /Users/agentxyz/Google Drive/Courses/STAT 385/su2017-priv/I
```

Unix Commands - cd - Change directory

```
cd ../ && pwd # Go one directory up
```

```
## /Users/agentxyz/Google Drive/Courses/STAT 385/su2017-priv/l
```

```
cd ~/ && pwd # Go to base directory
```

```
## /Users/agentxyz
```

Unix Commands - ls - List Files

```
ls ../                    # List files
```

```
## lec01
## lec02
```

```
ls -l ../ | grep lec01 # List files with lec01
```

```
## drwxr-xr-x  25 agentxyz  admin  850 Jun 12 14:02 lec01
```

Unix Commands - mkdir - Make Directory

- Use `mkdir` to create a new folder for a project.

```
mkdir test          # Make directory in `pwd`
```

- Adding the `-p` option allows for **all folders to be made** if not already present.

```
mkdir -p new/dir # The -p makes all directories
```

Unix Commands - rmdir - Remove directory

- Use `rmdir` to remove or delete a folder.

```
rmdir test          # Remove directory
```

- Including the `-p` option allows for all directory structures to be **removed**.

```
rmdir -p new/dir # The -p recursively removes
```

Useful Unix Commands - File Manipulation

Command	Description	Examples
`touch`	Make file	`touch file.R`
`vi`	Open text editor	`vi file.R`
`cat`	Display **All** of file	`cat file.R`
`chmod`	Set file permissions	`chmod 744 file.R`
`head`	Display *first* lines	`head file.R`
`tail`	Display *last* lines	`tail file.R`
`cp`	Copy file from x to y	`cp file1.R file2.R`
`mv`	Move (rename) file	`mv file_old.R file_new.R`
`rm`	Remove file	`rm file.R` **or** `rm file*.R`
`echo`	Display terminal variable	`echo $HOME`
`grep`	Regex find	`grep "toad"`

Unix Commands - touch - Touch

```
ls -l | grep "file.R" # File does not exist

# empty return
```

```
touch file.R            # Create File

ls -l | grep "file.R" # Check for existence
```

```
## -rw-r--r--  1 agentxyz  admin      0 Jun 13 11:57 file.R
```

Unix Commands - Search operators

Unix has the advantage of using regular expressions, regex, which we'll talk more about later, to search for files. Two operators to be cognisant about are:

- * - matches zero or more characters
- ? - matches any *one* character

```
ls -l *.R # Obtain any R file in the directory.
```

```
## -rw-r--r--  1 agentxyz  admin    0 Jun 13 11:57 file.R
## -rw-r--r--  1 agentxyz  admin    7 Jan 27 01:08 file_overwr
## -rw-------  1 agentxyz  admin  901 Jun 13 11:57 r_lec2.R
```

```
ls -l lec??.Rmd # Obtain any lec<xx>.Rmd file in the directory
```

```
## -rw-------  1 agentxyz  admin  22573 Jun 13 14:32 lec02.Rmd
```

Unix Commands - vi - File Editor in Terminal

```
vi file.R # Open file
```

- Navigating vi
  - Press I to insert new characters.
  - To save changes, press Esc and type :w
  - To exit, press Esc and type :q!
  - To do both at the same time use :wq!
- Resources:
  - **Interactive vim tutorial**
  - Try the **vim game** for practice
  - **vi Reference guide**

Note: vim is the sucessor to vi and still is applicable.

Unix Commands - Using redirection to write to file

Redirecting and *appending* output onto a file *avoids* the need for entering into an editor.

```
echo "line 1" >> file.R
echo "line 2" >> file.R
```

Unix Commands - cat - See file contents

```
cat file.R                    # Show file contents
```

```
## line 1
## line 2
```

Unix Commands - Using redirection to write to file

Using **heredoc format** enables the ability to write multiple lines simultaneously to the file.

```
cat << EOF >> file.R
line 3
line 4
EOF
```

What values are currently held by `file.R`?

Unix Commands - Redirection Redux Redux

Previously, we appended *onto* the file. In this case, the file will be overriden each time we write to it.

```
cp file.R file_overwrite.R          # Make a copy of file.R
echo "line 1" > file_overwrite.R
echo "line 2" > file_overwrite.R
```

- What is the initial state of `file_overwrite.R`?
- After the script runs, how does `file_overwrite.R` change?

Unix Commands - Summary of Redirection

- Note the following:
  - > outputs *and* overwrites the file
  - >> appends to a file
  - < reads input from file.

Unix Commands - File Permissions

- File permissions are a bit complicated but a necessary force.
- File permissions indicate whether someone can:

Type	Description	Value	Character
Execute	Run a file	1	`x`
Write	Save to a file	2	`w`
Read	See what a file contains.	4	`r`

Unix Commands - File Permissions for User Type

- Each type can be added together to customize the access level
  - For example: 7 would give all permissions, 5 gives only execute and read.
- There are **three** types of permissions that can be assigned:

Type	Description	Position	Character
User	Owner or user	First	`u`
Group	Those that belong to a group	Second	`g`
World	Everyone.	Third	`a`

Unix Commands - chmod - Set File Permissions

```
chmod 777 file.R        # User, Group, and World
                        # can execute, write, read

chmod 755 file.R        # Only User can execute, write, read
                        # Group and World can execute and read

chmod u+wxr file.R      # Only User can read, write, access
```

Unix Commands - head - See top content

```
head -2 file.R   # Show top 2 lines
```

```
## line 1
## line 2
```

- The -2 limits it to the **top** 2 observations

Unix Commands - `tail` - See bottom content

```
tail -1 file.R   # Show last line
```

```
## line 4
```

- The `-1` limits it to the **last** observation

Unix Commands - cp - Copy File

```
cp file.R file.R.bck # Create a back up
```

```
ls -l | grep ".bck"  # Check that it is there
```

```
## -rw-r--r--  1 agentxyz  admin     28 Jun 13 11:58 file.R.bc
```

- It is good practice to create `.bck` up files
- This is especially the case if you are working with configuration files (e.g. `.conf`)

Unix Commands - mv - Move File

```
mv file.R.bck file_in_use.R          # Rename file

mv file_in_use.R img/file_in_use.R # Move to new directory
```

- Moving a file is the *only* way to rename.

Unix Commands - rm - Remove file

```
rm file.R                  # Remove file
```

```
# Remove file in different directory
rm img/file_in_use.R
```

Unix Commands - echo - Display bash variables

```
samplevar="Hi stat385"     # Create a variable

echo $samplevar            # Print variable
```

```
## Hi stat385
```

Note the following:

- No space between variable, assignment operator, and value.
- The use of $ to refer to the variable in echo.