

Introduction to Terminal

Software Tools for Business Analytics: Lecture 1

Galit Lukin

Based on Slides by Jackie Baek

MIT

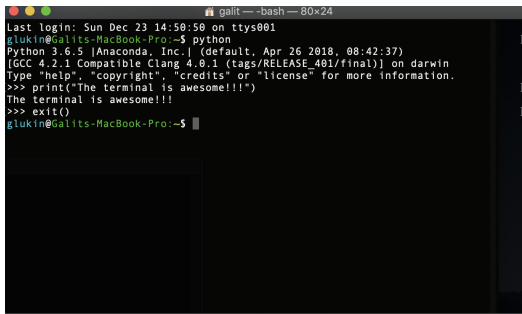
January 14th, 2019

What is the terminal?

- ▶ Console, Shell, Command line, Command prompt



What is the terminal?



```
galit ~ -bash — 80x24
Last login: Sun Dec 23 14:50:50 on ttys001
glukin@Galits-MacBook-Pro:~$ python
Python 3.6.5 |Anaconda, Inc.| (default, Apr 26 2018, 08:42:37)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("The terminal is awesome!!!")
The terminal is awesome!!!
>>> exit()
glukin@Galits-MacBook-Pro:~$
```

- ▶ The terminal is a text-based interface to interact with the computer.
- ▶ For example, it can replace the use of the file system and the use of IDEs

Example

- ▶ Say you want to delete all files in a directory that end with .csv

```
$ rm *.csv
```

- ▶ Or change their location to a folder for outputs

```
$ mv *.csv ../OutputFolder
```

- ▶ This is possible to do without the terminal, but it requires much more effort.

Why should I learn it?

- ▶ You can do almost everything using just the terminal.
- ▶ It can do many tasks faster than using a graphic interface.
- ▶ You can simultaneously run different simulations with different parameters.
- ▶ Using the terminal is sometimes the only option (e.g. accessing a client's server using SSH).
- ▶ The terminal is universal.

Use case: running code

- ▶ Without the terminal, you need to install an IDE (Integrated development environment) for every programming language.
 - ▶ e.g. RStudio for R
 - ▶ e.g. PyCharm for Python

Use case: running code

- ▶ Without the terminal, you need to install an IDE (Integrated development environment) for every programming language.
 - ▶ e.g. RStudio for R
 - ▶ e.g. PyCharm for Python
- ▶ Useful to use the terminal instead of IDEs when:

Use case: running code

- ▶ Without the terminal, you need to install an IDE (Integrated development environment) for every programming language.
 - ▶ e.g. RStudio for R
 - ▶ e.g. PyCharm for Python
- ▶ Useful to use the terminal instead of IDEs when:
 - ▶ You use more than one programming language.

```
$ python process_stuff.py
```

```
$ R make_plots.R
```


Use case: running code

- ▶ Without the terminal, you need to install an IDE (Integrated development environment) for every programming language.
 - ▶ e.g. RStudio for R
 - ▶ e.g. PyCharm for Python
- ▶ Useful to use the terminal instead of IDEs when:
 - ▶ You use more than one programming language.

```
$ python process_stuff.py
```

```
$ R make_plots.R
```

- ▶ You want to chain commands together.
 - ▶ The following command will execute the command on the right if and only if the command on the left succeeded.

```
$ python process_stuff.py && R make_plots.R
```

Use case: running code

- ▶ Without the terminal, you need to install an IDE (Integrated development environment) for every programming language.
 - ▶ e.g. RStudio for R
 - ▶ e.g. PyCharm for Python
- ▶ Useful to use the terminal instead of IDEs when:
 - ▶ You use more than one programming language.

```
$ python process_stuff.py
```

```
$ R make_plots.R
```

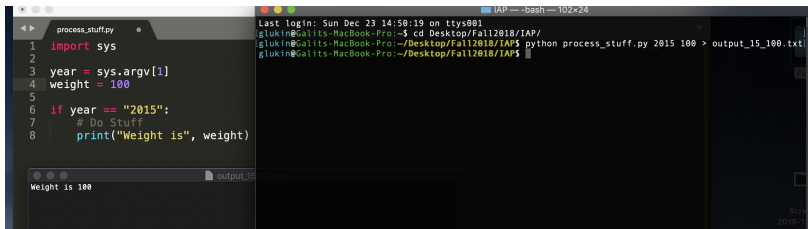
- ▶ You want to chain commands together.
 - ▶ The following command will execute the command on the right if and only if the command on the left succeeded.

```
$ python process_stuff.py && R make_plots.R
```

- ▶ You want to run a script with different parameters and different output files.

```
$ python process_stuff.py 2015 100 > output_15_100.txt
```

Use case: running a script with parameters and output files



The screenshot displays a development environment with a code editor on the left and a terminal on the right. The code editor shows a Python script named `process_stuff.py` with the following content:

```
1 import sys
2
3 year = sys.argv[1]
4 weight = 100
5
6 if year == "2015":
7     # Do Stuff
8     print("Weight is", weight)
```

The terminal window, titled `IAP - bash - 102x24`, shows the execution of the script. The user navigates to the directory `~/Desktop/Fall2018/IAP/` and runs the command `python process_stuff.py 2015 100 > output_15_100.txt`. The output of the script, `Weight is 100`, is visible in the terminal's output pane.

Use case: learning a language

- ▶ Instead of constantly Googling or running scripts that fail
- ▶ Have a separate terminal open to test your syntax!

A terminal window titled "galit — julia — 80x24". The prompt shows the user is logged as glukin@Galits-MacBook-Pro-~\$ and has entered 'julia'. On the left side of the terminal, there are several small colored circles (blue, green, red) arranged vertically. Below them is a diagram consisting of dashed boxes connected by arrows, representing some form of state transition or graph. To the right of this diagram, there is introductory text about Julia: "A fresh approach to technical computing", documentation link "[https://docs.julialang.org](\"https://docs.julialang.org\")", version information "Version 0.6.4 (2018-07-09 19:09 UTC)", official website "[http://julialang.org/](\"http://julialang.org/\")", and platform identifier "x86_64-apple-darwin14.5.0". The main part of the terminal displays a series of commands and their outputs:
`[julia> my_vec = []
0-element Array{Any,1}`
`[julia> my_vec.append(1)
ERROR: type Array has no field append`
`[julia> my_vec.push!(1)
ERROR: type Array has no field push!`
`[julia> push!(my_vec,1)`
`1-element Array{Any,1}:
1`
The final prompt `julia>` is followed by a cursor icon.

Terminal Basics

- ▶ We will be using a **shell** called **bash**: a program that interprets and processes the commands you input into the terminal.
- ▶ The shell is always in a **working directory**.
- ▶ A typical command looks like:

```
$ command <argument1> <argument2> ...
```

Basic navigation commands

pwd: prints working directory.

```
$ pwd  
/Users/galit
```

Basic navigation commands

pwd: prints working directory.

```
$ pwd  
/Users/galit
```

ls: lists directory contents.

```
$ ls  
Applications      Movies  
Desktop           Music  
Documents         Pictures
```

Basic navigation commands

pwd: prints working directory.

```
$ pwd  
/Users/galit
```

ls: lists directory contents.

```
$ ls  
Applications                Movies  
Desktop                     Music  
Documents                   Pictures
```

cd <directory>: change working directory to new directory.

```
$ cd Desktop/Fall2018  
$ pwd  
/Users/galit/Desktop/Fall2018
```


Basic navigation commands

pwd: prints working directory.

```
$ pwd  
/Users/galit
```

ls: lists directory contents.

```
$ ls  
Applications                Movies  
Desktop                     Music  
Documents                   Pictures
```

cd <**directory**>: change working directory to new directory.

```
$ cd Desktop/Fall2018  
$ pwd  
/Users/galit/Desktop/Fall2018
```

open <**filename**>: opens the file - analogous to double-clicking.

```
$ open FallRegistration.pdf
```

Use tab, arrow keys and file path shortcuts

- ▶ Use **tab** to autocomplete *commands* and *file paths*.
- ▶ Use ↑ and ↓ arrow keys to navigate through your command history.

Use tab, arrow keys and file path shortcuts

- ▶ Use **tab** to autocomplete *commands* and *file paths*.
- ▶ Use ↑ and ↓ arrow keys to navigate through your command history.
- ▶ . is current directory.

```
$ open ../FallRegistration.pdf
```

- ▶ .. is parent directory.

```
$ cd IAP #Fall2018 is the parent directory of IAP  
$ open ../FallRegistration.pdf
```

Use tab, arrow keys and file path shortcuts

- ▶ Use **tab** to autocomplete *commands* and *file paths*.
- ▶ Use ↑ and ↓ arrow keys to navigate through your command history.

- ▶ . is current directory.

```
$ open ./FallRegistration.pdf
```

- ▶ .. is parent directory.

```
$ cd IAP #Fall2018 is the parent directory of IAP  
$ open ../FallRegistration.pdf
```

- ▶ ~ is home.
 - ▶ expands to `/Users/<username>` (or wherever *home* is on that machine).
 - ▶ `~/Documents` → `/Users/galit/Documents`
 - ▶ The command **cd** (without any arguments) takes you to ~.

What is a file?

- ▶ A file is a container of data (0's and 1's).

What is a file?

- ▶ A file is a container of data (0's and 1's).
- ▶ A file name usually has an **extension** (e.g. .pdf, .doc, .csv), but these are just conventions.

What is a file?

- ▶ A file is a container of data (0's and 1's).
- ▶ A file name usually has an **extension** (e.g. .pdf, .doc, .csv), but these are just conventions.
- ▶ A file is contained in a **directory** (folder). Files within the same directory have unique names.

What is a file?

- ▶ A file is a container of data (0's and 1's).
- ▶ A file name usually has an **extension** (e.g. .pdf, .doc, .csv), but these are just conventions.
- ▶ A file is contained in a **directory** (folder). Files within the same directory have unique names.
- ▶ Every file and directory has a unique location in the file system, called a **path**.
 - ▶ **Absolute path**:
/Users/galit/Desktop/Fall2018/FallRegistration.pdf
 - ▶ **Relative path** (if my current working directory is */Users/galit/Desktop*): *Fall2018/FallRegistration.pdf*

Working with files

mkdir *directory_name*: create a new directory.

```
$ mkdir new_directory
```

Working with files

mkdir *directory_name*: create a new directory.

```
$ mkdir new_directory
```

touch *file*: create an empty file.

rm *file*: delete a file (**Careful!** Can't be undone!)

```
$ touch brand_new_file.txt
```

```
$ rm brand_new_file.txt
```

Working with files

mkdir *directory_name*: create a new directory.

```
$ mkdir new_directory
```

touch *file*: create an empty file.

rm *file*: delete a file (**Careful!** Can't be undone!)

```
$ touch brand_new_file.txt
```

```
$ rm brand_new_file.txt
```

nano *file*: edit contents of a file (many other editors exist).

```
$ nano helloworld.txt
```

Working with files

mkdir *directory_name*: create a new directory.

```
$ mkdir new_directory
```

touch *file*: create an empty file.

rm *file*: delete a file (**Careful!** Can't be undone!)

```
$ touch brand_new_file.txt
```

```
$ rm brand_new_file.txt
```

nano *file*: edit contents of a file (many other editors exist).

```
$ nano helloworld.txt
```

cat *file*: prints contents of a file.

```
$ cat helloworld.txt
```

```
Hello, World!
```

Working with files

mkdir *directory_name*: create a new directory.

```
$ mkdir new_directory
```

touch *file*: create an empty file.

rm *file*: delete a file (**Careful!** Can't be undone!)

```
$ touch brand_new_file.txt
```

```
$ rm brand_new_file.txt
```

nano *file*: edit contents of a file (many other editors exist).

```
$ nano helloworld.txt
```

cat *file*: prints contents of a file.

```
$ cat helloworld.txt
```

```
Hello, World!
```

cp *source target*: copy.

mv *source target*: move/rename.

```
$ cp helloworld.txt helloworld_copy.txt
```

```
$ mv helloworld.txt goodbyeworld.txt
```

Hidden Files

- ▶ Files that start with a dot (.) are called **hidden** files.
- ▶ Used for storing preferences, config, settings.
- ▶ Use `ls -a` to list all files.

```
$ ls
```

```
github_notes.md  presentation  scripts
```

```
$ ls -a
```

```
.                .git          github_notes.md  scripts
..               .gitignore    presentation
```

~/.bashrc, ~/.bash_profile

- ▶ There is a hidden file in ~ called .bashrc or .bash_profile.
- ▶ This file is a bash script that runs at the beginning of each session (i.e. when you open the terminal).

~/.bashrc, ~/.bash_profile

- ▶ There is a hidden file in ~ called .bashrc or .bash_profile.
- ▶ This file is a bash script that runs at the beginning of each session (i.e. when you open the terminal).
- ▶ This file can be used to set variables or to declare **aliases**.
 - ▶ What is the difference?

~/.bashrc, ~/.bash_profile

- ▶ There is a hidden file in ~ called .bashrc or .bash_profile.
- ▶ This file is a bash script that runs at the beginning of each session (i.e. when you open the terminal).
- ▶ This file can be used to set variables or to declare **aliases**.
 - ▶ What is the difference?
 - ▶ Variables can be used anywhere in a command line (e.g. as parts of program arguments)
 - ▶ Aliases can only be used as the names of programs to run (e.g. cd, ssh, mkdir)

- ▶ **alias** *new_command=command*

```
$ alias fall2018="cd ~/Desktop/Fall2018"
```

```
$ alias athena="ssh glukin@athena.dialup.mit.edu"
```

- ▶ **PATH**=*path:\$PATH*

```
$ PATH="/Applications/anaconda3/bin:$PATH"
```

Redirection

> redirects output to a file, *overwriting* if file already exists.

```
$ ls > out.txt
```

>> redirects output to a file, *appending* if file already exists.

```
$ python fetch_data.py >> output.csv
```

Redirection

> redirects output to a file, *overwriting* if file already exists.

```
$ ls > out.txt
```

>> redirects output to a file, *appending* if file already exists.

```
$ python fetch_data.py >> output.csv
```

< uses contents of file as STDIN (standard input) to the command.

```
$ python process_stuff.py < input.txt
```

Secure Shell (SSH)

- ▶ Sometimes we need to work on a remote machine.
 - ▶ We need more computing power than just our local machine.
 - ▶ We need to access data from a client's server.
- ▶ Can use SSH to securely access the terminal for the remote machine.

Secure Shell (SSH)

- ▶ Sometimes we need to work on a remote machine.
 - ▶ We need more computing power than just our local machine.
 - ▶ We need to access data from a client's server.
- ▶ Can use SSH to securely access the terminal for the remote machine.

```
$ ssh glukin@athena.dialup.mit.edu
```

or

```
$ athena
```

Secure Shell (SSH)

- ▶ Sometimes we need to work on a remote machine.
 - ▶ We need more computing power than just our local machine.
 - ▶ We need to access data from a client's server.
- ▶ Can use SSH to securely access the terminal for the remote machine.

```
$ ssh glukin@athena.dialup.mit.edu
```

```
or
```

```
$ athena
```

```
Password:
```

Secure Shell (SSH)

- ▶ Sometimes we need to work on a remote machine.
 - ▶ We need more computing power than just our local machine.
 - ▶ We need to access data from a client's server.
- ▶ Can use SSH to securely access the terminal for the remote machine.

```
$ ssh glukin@athena.dialup.mit.edu  
or  
$ athena
```

Password:

```
Welcome to Ubuntu 14.04.5 LTS  
...  
Last login:  Sun Dec 23 10:56:22 2018 ....  
  
glukin@buzzword-bingo:~$
```

Use *logout* to exit SSH session.

Secure Copy (scp)

Can transfer files between local and remote machines using the **scp** command on your local machine.

Move *my_file.txt* from local machine to remote home directory.

```
$ scp my_file.txt glukin@athena.dialup.mit.edu:~
```

Move *remote_file.txt* from remote to local machine.

```
$ scp glukin@athena.dialup.mit.edu:~/remote_file.txt .
```


Simple Pattern Matching (Globbing)

- ▶ Match [multiple] filenames with wildcard characters.
- ▶ Similar to *regular expressions*, but slightly different syntax.

Simple Pattern Matching (Globbing)

- ▶ Match [multiple] filenames with wildcard characters.
- ▶ Similar to *regular expressions*, but slightly different syntax.

Example:

```
$ ls
```

```
a1.txt    a2.pdf    apple.txt  bar.pdf
```

Simple Pattern Matching (Globbing)

- ▶ Match [multiple] filenames with wildcard characters.
- ▶ Similar to *regular expressions*, but slightly different syntax.

Example:

```
$ ls
```

```
a1.txt      a2.pdf      apple.txt    bar.pdf
```

```
$ ls a*
```

```
a1.txt      a2.pdf      apple.txt
```

Simple Pattern Matching (Globbing)

- ▶ Match [multiple] filenames with wildcard characters.
- ▶ Similar to *regular expressions*, but slightly different syntax.

Example:

```
$ ls
```

```
a1.txt      a2.pdf      apple.txt    bar.pdf
```

```
$ ls a*
```

```
a1.txt      a2.pdf      apple.txt
```

```
$ ls *.pdf
```

```
a2.pdf      bar.pdf
```

Simple Pattern Matching (Globbing)

- ▶ Match [multiple] filenames with wildcard characters.
- ▶ Similar to *regular expressions*, but slightly different syntax.

Example:

```
$ ls
```

```
a1.txt      a2.pdf      apple.txt    bar.pdf
```

```
$ ls a*
```

```
a1.txt      a2.pdf      apple.txt
```

```
$ ls *.pdf
```

```
a2.pdf      bar.pdf
```

```
$ ls a[0-9]*
```

```
a1.txt      a2.pdf
```

Simple Pattern Matching (Globbing)

Wildcard	Description	Example	Matches
*	matches any number of any characters including none	Law*	Law , Laws , Or Lawyer
		Law	Law , GrokLaw , Or Lawyer .
?	matches any single character	?at	Cat , cat , Bat or bat
[abc]	matches one character given in the bracket	[CB]at	Cat or Bat
[a-z]	matches one character from the range given in the bracket	Letter[0-9]	Letter0 , Letter1 , Letter2 up to Letter9

Source: Wikipedia

Simple Pattern Matching (Globbing)

Wildcard	Description	Example	Matches
*	matches any number of any characters including none	Law*	Law , Laws , Or Lawyer
		Law	Law , GrokLaw , Or Lawyer .
?	matches any single character	?at	Cat , cat , Bat or bat
[abc]	matches one character given in the bracket	[CB]at	Cat or Bat
[a-z]	matches one character from the range given in the bracket	Letter[0-9]	Letter0 , Letter1 , Letter2 up to Letter9

Source: Wikipedia

Remove all files that end with .pyc

```
$ rm *.pyc
```

Simple Pattern Matching (Globbing)

Wildcard	Description	Example	Matches
*	matches any number of any characters including none	Law*	Law , Laws , Or Lawyer
		Law	Law , GrokLaw , Or Lawyer .
?	matches any single character	?at	Cat , cat , Bat or bat
[abc]	matches one character given in the bracket	[CB]at	Cat or Bat
[a-z]	matches one character from the range given in the bracket	Letter[0-9]	Letter0 , Letter1 , Letter2 up to Letter9

Source: Wikipedia

Remove all files that end with .pyc

```
$ rm *.pyc
```

Copy all files that has "dog" in its name to the *animal/* directory.

```
$ cp *dog* animal/
```


How bash works

- ▶ Bash is a programming language.
 - ▶ Can set variables, use for loops, if statements, comments, etc.

How bash works

- ▶ Bash is a programming language.
 - ▶ Can set variables, use for loops, if statements, comments, etc.
- ▶ There are several special "environment" variables (i.e. \$PATH, \$HOME, \$USER, etc.) that many programs rely on.

How bash works

What happens when you type in a command, say *pwd*?

How bash works

What happens when you type in a command, say *pwd*?

- ▶ Bash runs the program called *pwd*.
- ▶ Where is this program?
 - ▶ Usually under a directory called *bin*, which stands for *binary*.

How bash works

What happens when you type in a command, say *pwd*?

- ▶ Bash runs the program called *pwd*.
- ▶ Where is this program?
 - ▶ Usually under a directory called *bin*, which stands for *binary*.
- ▶ When you type in a command, bash looks for a program with that name under the directories listed in the *\$PATH* environment variable.

How bash works

What happens when you type in a command, say *pwd*?

- ▶ Bash runs the program called *pwd*.
- ▶ Where is this program?
 - ▶ Usually under a directory called *bin*, which stands for *binary*.
- ▶ When you type in a command, bash looks for a program with that name under the directories listed in the *\$PATH* environment variable.

```
$ echo $PATH
```

```
/Applications/anaconda3/bin:  
/Library/Frameworks/Python.framework/Versions/3.6/bin:  
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:  
/Library/TeX/texbin
```

- ▶ *\$PATH* contains is liist of directories separated by :
- ▶ Bash looks into each of these directories to look for the program *pwd*.

Common Error: Command not found

You installed a new software (e.g. TeX), but as soon as you try to run it, you get this error:

```
$ pdflatex
```

```
Error: pdflatex: command not found.
```

Common Error: Command not found

You installed a new software (e.g. TeX), but as soon as you try to run it, you get this error:

```
$ pdflatex
```

```
Error: pdflatex: command not found.
```

- ▶ This means that bash cannot find the program 'pdflatex' in the *\$PATH* variable.

Common Error: Command not found

You installed a new software (e.g. TeX), but as soon as you try to run it, you get this error:

```
$ pdflatex
```

```
Error: pdflatex: command not found.
```

- ▶ This means that bash cannot find the program 'pdflatex' in the *\$PATH* variable.
- ▶ **Solution:** Find where you installed TeX, find the directory with the binary files (usually a directory called *bin*), and add the directory to *\$PATH*.

Common Error: Command not found

You installed a new software (e.g. TeX), but as soon as you try to run it, you get this error:

```
$ pdflatex  
Error: pdflatex: command not found.
```

- ▶ This means that bash cannot find the program 'pdflatex' in the *\$PATH* variable.
- ▶ **Solution:** Find where you installed TeX, find the directory with the binary files (usually a directory called *bin*), and add the directory to *\$PATH*.
- ▶ Add the following to your *~/.bash_profile*:

```
PATH="$PATH:/Library/TeX/Distributions/Programs/texbin"  
export PATH
```
- ▶ The export command allows a child process to inherit all marked variables

Key Takeaways

- ▶ Basic commands: ls, cd, pwd, cat, cp, mv, rm, mkdir

Key Takeaways

- ▶ Basic commands: ls, cd, pwd, cat, cp, mv, rm, mkdir
- ▶ No need to use IDEs!

Key Takeaways

- ▶ Basic commands: ls, cd, pwd, cat, cp, mv, rm, mkdir
- ▶ No need to use IDEs!
- ▶ Google is your friend.

Key Takeaways

- ▶ Basic commands: `ls`, `cd`, `pwd`, `cat`, `cp`, `mv`, `rm`, `mkdir`
- ▶ No need to use IDEs!
- ▶ Google is your friend.
- ▶ So is *tab* for autocomplete, *arrow keys* for history.

Key Takeaways

- ▶ Basic commands: `ls`, `cd`, `pwd`, `cat`, `cp`, `mv`, `rm`, `mkdir`
- ▶ No need to use IDEs!
- ▶ Google is your friend.
- ▶ So is *tab* for autocomplete, *arrow keys* for history.
- ▶ Be careful with *rm*.

Key Takeaways

- ▶ Basic commands: `ls`, `cd`, `pwd`, `cat`, `cp`, `mv`, `rm`, `mkdir`
- ▶ No need to use IDEs!
- ▶ Google is your friend.
- ▶ So is *tab* for autocomplete, *arrow keys* for history.
- ▶ Be careful with *rm*.
- ▶ Getting comfortable with the terminal can be daunting at first, but it has the potential to greatly boost your efficiency!

Thank you!