

Unix navigation tutorial

Experiential Data science for Undergraduate Cross-disciplinary Education

Dr. Kim Dill-McFarland, U. of British Columbia

Contents

Unix navigation tutorial	1
Learning objectives	1
Accessing the command line	1
A first view of the terminal	2
Navigating your computer	2

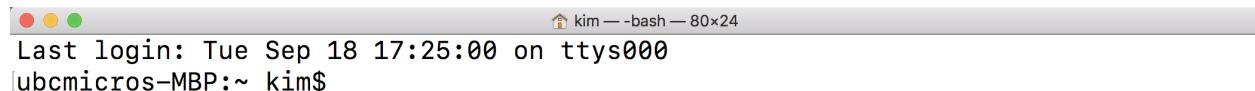
Unix navigation tutorial

Learning objectives

- Define parts of the terminal
- Use Unix commands to navigate your computer including pwd, ls, man/help, and cd

Accessing the command line

To begin, open a command line window on your computer. This is also called the shell, terminal, or bash. This will be the built in command line for Mac machines, which can be found by searching “terminal” within your Applications directory.



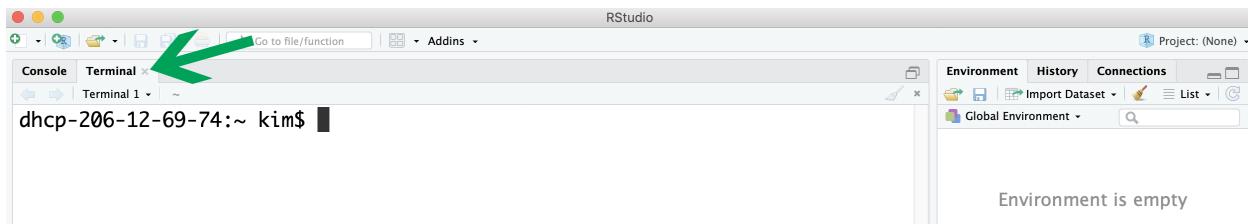
```
kim — bash — 80x24
Last login: Tue Sep 18 17:25:00 on ttys000
[ubcmicros-MBP:~ kim$
```

or Git Bash for Windows machines. This terminal is installed as part of Git and can be found by searching “Git” in your start menu. Be sure to open Git Bash and *not Git CMD or GUI*.



```
MINGW64:/c/Users/kdill
kdill@DESKTOP-2ROMA1P MINGW64 ~
$
```

or the terminal integrated into RStudio.



Throughout this tutorial, you will generally see the R terminal as the example output. If something looks significantly different, we will also show that window in the other terminal programs.

A first view of the terminal

Command prompt

Notice the \$ symbol in the above windows. This is the command prompt. When you execute (*i.e.* run) a command, the prompt will not appear on a new line until your computer has completed the previous function(s). Thus, when you see the \$, you know that the terminal is ready for you to input your next command.

Identifying information

A new terminal window will also let you know some information about the computer you are working on. This can include the

- **Last login:** The Mac terminal shows the last time someone opened a terminal and the identifying number of that terminal (like ttys000). Unless you're worried about someone hacking into your machine, you don't need to worry about this info. In fact, Git Bash on Windows does not even show it.
- **Unix version:** Since Git is not pre-installed as part of Windows, Git Bash shows the the version of Unix associated with the suite of Git tools you downloaded (MINGW64 for a 64-bit system).
- **Computer name/ID:** Both terminals tell you the name of the computer you are working on. Mac shows this name before a colon (ubcmicro-MBP) while Git Bash shows it after a @ symbol (DESKTOP-BB44ULA). The name you see will be specific to the computer you are working on.
- **Home directory:** Both terminals open in what is called the home directory. This is generally your user directory like `kim` or `kdill` in the examples (more on this under the `pwd` command).

All of these pieces of information help you to know what computer you're working on and where on that computer you're working. This may not seem useful as you work on a computer sitting right in front of you. However, it is extremely helpful when remotely accessing other machines and/or using multiple terminal windows at once.

Navigating your computer

`pwd`

When you open a new terminal, you will automatically be in your `home` directory. To find out where this is, use `pwd` which stands for **print working directory**. This tells you where your terminal is currently pointing.

```
pwd
```

```
## /Users/kim
```

You can see here than I am in my home user directory `kim`.

`ls`

To view all the files and sub-directories in the current directory, use `ls` for **list**.

`ls`

```
## Applications
## Desktop
## Documents
## Documents_large
## Downloads
## GitHub
## Google Drive
## Library
## Movies
## Music
## Pictures
## Public
## Video
## VirtualBox VMs
## test.txt
```

You can see that my home directory has a number of things that all Macs will have (*e.g.* Applications, Desktop, etc.) as well as a number of things which are unique to my file system (*e.g.* Documents_large, GitHub, etc.) Your list will be unique to your computer.

`ls` (Git Bash)

For those working in Git Bash, you will automatically see some additional information with `ls`.

```
kdi11@DESKTOP-BB44ULA MINGW64 ~
$ ls
'3D Objects'/
'AppData'/
'Application Data'@
'Contacts'/
'Cookies'@
'Desktop'/
'Documents'/
'Downloads'/
'Favorites'/
'Links'/
'Local Settings'@
'MicrosoftEdgeBackups'/
'Music'/
'My Documents'@
'NetHood'@
'NTUSER.DAT'
```

Git Bash colors files by type as well as includes indicators at the end of some types. Folders are blue and end in `/`, symbolic links (like shortcuts) are teal and end in `@`, files are grey, etc.

`ls` options

You can ask for these indicators in the Mac terminal by adding modifiers to the `ls` command. You do this with `-` after the command. For example, if you want to show all of the file type indicators, you would use `ls -F`.

```
ls -F

## Applications/
## Desktop/
## Documents/
## Documents_large/
## Downloads/
## GitHub/
## Google Drive/
## Library/
## Movies/
## Music/
## Pictures/
## Public/
## Video/
## VirtualBox VMs/
## test.txt
```

Now you will see that everything in my home directory are themselves directories ending in /.

man

Most commands can be modified with text following a - as in `ls -F`. You can learn more about these modifications by looking up the **manual** of any function with `man`. For example, in R or Mac, if you look-up `pwd` with `man pwd`, you will see the following.

```
man pwd
```

```
##
##          PWD(1)           BSD General Commands Manual          PWD(1)
##
## NAME
##       pwd -- return working directory name
##
## SYNOPSIS
##       pwd [-L | -P]
##
## DESCRIPTION
##       The pwd utility writes the absolute pathname of the current working
##       directory to the standard output.
##
##       Some shells may provide a builtin pwd command which is similar or identi-
##       cal to this utility. Consult the builtin(1) manual page.
##
##       The options are as follows:
##
##       -L      Display the logical current working directory.
##
##       -P      Display the physical current working directory (all symbolic
##              links resolved).
##
##       If no options are specified, the -L option is assumed.
##
## ENVIRONMENT
##       Environment variables used by pwd:
```

```

##          PWD  Logical current working directory.
##
## EXIT STATUS
##          The pwd utility exits 0 on success, and >0 if an error occurs.
##
## SEE ALSO
##          builtin(1), cd(1), csh(1), sh(1), getcwd(3)
##
## STANDARDS
##          The pwd utility conforms to IEEE Std 1003.1-2001 (`POSIX.1').
##
## BUGS
##          In csh(1) the command dirs is always faster because it is built into that
##          shell.  However, it can give a different answer in the rare case that the
##          current directory or a containing directory was moved after the shell
##          descended into it.
##
##          The -L option does not work unless the PWD environment variable is
##          exported by the shell.
##
## BSD           April 12, 2003           BSD

```

This is a text documents with information on every possible modifier of the `pwd` function. You can **exit the manual** back to the regular terminal with `q` for 'quit'.

help (Git Bash)

Similarly, you can look up the **manual** in a Git Bash terminal with `--help`. For example, if you look-up `ls` with `pwd --help`, you will see the following.

```

kd11@DESKTOP-BB44ULA MINGW64 ~
$ pwd --help
pwd: pwd [-LPW]
      Print the name of the current working directory.

Options:
  -L      print the value of $PWD if it names the current working
         directory
  -P      print the physical directory, without any symbolic links
  -W      print the Win32 value of the physical directory

By default, `pwd' behaves as if '-L' were specified.

Exit Status:
Returns 0 unless an invalid option is given or the current directory
cannot be read.

```

cd

So now that you've seen what directories exist in your home directory, you can move into another directory with `cd` for **change directory**. Let's move into the Desktop since everyone will have this directory. Note that the command line is *case sensitive* so `Desktop` $=/=$ `desktop`.

```

cd Desktop
pwd

```

```
## /Users/kim/Desktop
```

You can confirm that you've actually moved by using `pwd` again.

Shortcuts

You now have a couple of options for getting back out of the Desktop.

1. You can go **up one directory** using `cd ..`
 - In this case, you would go back to your home directory since this is one up from the Desktop directory.
 - You can repeat this as many times as you want and continue to go up one directory at a time as shown.

```
pwd  
  
cd ..  
pwd  
  
cd ..  
pwd  
  
## /Users/kim/Desktop  
## /Users/kim  
## /Users
```

2. You can go straight to the home directory from anywhere on your computer using `cd ~`.
 - The `~` is shorthand for your **home directory**.

```
pwd  
  
cd ~  
pwd  
  
## /Users/kim/Desktop  
## /Users/kim
```

No matter how many directories up or down you currently are, `~` will take you to your home directory.

Tab completion

As file names or paths get longer, it becomes cumbersome to type them out. To make your life easier and prevent typos, you can use tab completion instead. This will fill in the rest of a directory or file name *if it is unique*.

For example, let's look at my home directory again.

```
ls  
  
## Applications  
## Desktop  
## Documents  
## Documents_large  
## Downloads  
## GitHub  
## Google Drive  
## Library  
## Movies  
## Music
```

```
## Pictures  
## Public  
## Video  
## VirtualBox VMs  
## test.txt
```

If I were to type `Desk` and then use the [tab], the terminal would automatically complete `Desktop/` because that is the only directory in the current folder that starts with `Desk`.

In contrast, if I tried `Doc` [tab], the terminal would not know if I meant `Documents` or `Documents_large`, which both exist in my home `kim` directory. So, depending on the terminal version you have, it would either fill in as much as it could (*e.g.* `Documents` since both directory names start with that) or fail to fill anything in.

Remember to give the terminal at least 3 letters/numbers before trying [tab] and to use file/directory names that exist on your computer.

Exiting command line

This concludes the main commands used to navigate a computer. Now you *could* end your terminal session by closing the window with the [X]. However, this is not best practices and could cause issues if you are remotely logged into another machine.

It is best to officially exit the session with `exit`. This will save your history and safely disconnect you. Git Bash will then automatically close while the Mac or R terminal will remain open but disconnected.

```
exit
```
