Working with Git Bash

JSC270 Lab 4: Part I

Matthew Edwards

mr.edwards@utoronto.ca

Feb 3, 2021

Last Week: Git

Recall last week, we learned about **version control**

- **Git** is a version control system used to track changes in sets of files (called repositories, or *repos*)
- It was originally designed for computer programmers to collaborate during software development
- Today, almost all data science teams (in both academia and industry) use it to track their work
- The system includes a small (but very powerful) set of commands for editing and tracking files, including:
  - `push`
  - `pull`
  - `add`
  - `commit`
  - `clone`
  - `branch`

Last Week: GitHub

If the goal is collaboration, these file collections (repos) should be stored somewhere easily accessible

- **GitHub** is an internet hosting service designed to store Git repositories. It allows for easy interaction with Git
- It is not the only service that does this (e.g. GitLab, BitBucket, SorceForge), but it is by far the most popular (and free)
- Git and GitHub can track multiple branches (versions) of the same repository, so **contributors can make changes in parallel** without affecting the `main/master` branch
  - This is particularly useful if you're experimenting (ie you're not sure if your changes will actually work)

A Basic Git Workflow

Here is a basic step-by-step process for managing repos:

1. You have a new repo you'd like to work with. Either you've created it yourself, or it is someone else's **public** repository.
   - Recall from last week: Only you (the repo creator) and contributors you allow can see a private repo.
2. You **Fork** or **Clone** that repo to your local machine (e.g into your local `gitrepos` folder). More on this later.
3. You edit files in the local repo (or create new ones). If you're experimenting, you would first create a new **branch**, so your changes do not affect `main`.

A Basic Git Workflow

4. You submit any changes you've made so they can be tracked by Git. This is a two-part process:

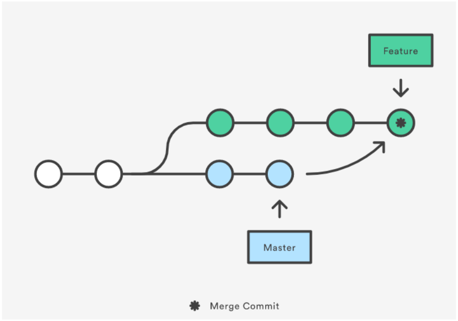
   i) **Use the `add` command** to move changes to the staging area. This tells Git that you have changes to be included in your next commit.

   ii) **Use the `commit` command** to save these changes to your local repo. This effectively takes a 'snapshot' of your files.

Note:
Committed snapshots are safe - Git will never change them unless you tell it to.

Every commit you make is recorded and tracked, so you may revert back to past versions of your project at any time.

A Basic Git Workflow



5. (Optional) If you were experimenting on a different branch, you can `merge` your newer version back into the main branch.

6. You use the `push` command to push your commit(s) from your local repo to the remote repo (on GitHub). Again, more on this later.

Source:https://www.atlassian.com/git/tutorials/merging-vs-rebasing

A Basic Git Workflow

7. If other contributors are following a similar process, it's likely they will `push` changes to the remote (GitHub) repo that you do not have locally. **Use the *pull* command** to to update your local repository to the most current version.

Note:
In this course, you will be the only ones editing your assignment files (although we need contributor access to read them), so steps 5 and 7 may not be necessary.

But in practice, there could be many contributors, and the remote repo might be ahead of (upstream from) your local version. So a good rule is to **always `pull` before you make changes**.

Question Time

Any questions about Git or GitHub (or about anything else so far)?

What is Bash?

So what does any of this have to do with Bash?

- A **command line interface** is an interpreter that processes text commands to allow interaction with your computer's operating system.
- The **B**orn-**a**gain **Sh**ell, or **Bash**, invented in 1989, is one of the most popular CLIs available
- Bash is designed to work seamlessly with Git, but also makes it very easy to navigate through the files on your computer
- **Every step in the workflow above can be done in Bash**

Installing Bash

Mac Users: Good News! Apple computers use a version of Bash as the default CLI.

- To open the shell, simply use `Command + Spacebar` to launch `Spotlight`, then search **Terminal** and double-click the search result
- Alternatively you could click through `Applications -> Utilities` and find **Terminal** in there

Installing Bash

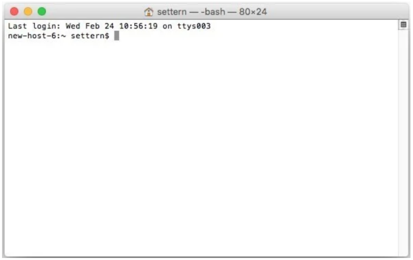
Windows Users: Unfortunately Bash is not the default, so we'll have to install it separately

1. Go to the install page: `https://gitforwindows.org/`
2. Click the 'Download' button
3. Once downloaded, run the **.exe** file
4. "Do you want to allow this app to make changes to your device" -> **Yes**
5. Use the default settings. Keep clicking **Yes**, and then eventually, click **Finish**

Installing Bash

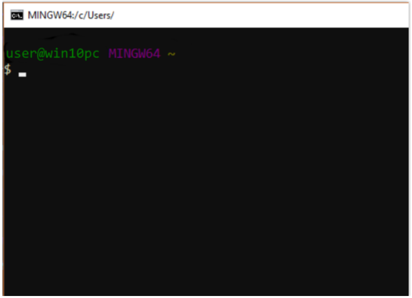
For Windows, the application is called **Git Bash**.

- You can open this by searching 'Git Bash' in the Windows Start menu



```
Last login: Wed Feb 24 10:56:19 on ttys003
new-host-6:~ settern$
```

Mac Shell



```
user@win10pc MINGW64 ~
$ _
```

Windows Shell

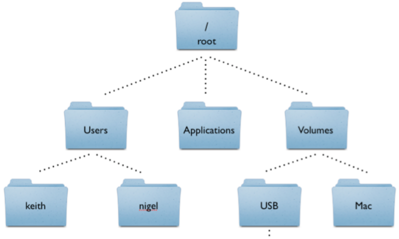
Exercise 1

- Windows users, go ahead and install Git Bash using the instructions above. Then open the Bash shell
- Mac Users (or any Windows users already familiar with Bash) can just open the terminal.
- Please type **Done** into the Zoom chat once you've completed this, so I know when to move forward.

Navigating Your File System

- Your computer stores files in a **tree structure**
- the top of the tree is called the **root**
- Individual files are grouped together in folders, also known as **directories**
- Some directories can be made into git repos

*When you open a new bash shell, you are automatically at the root of your file tree

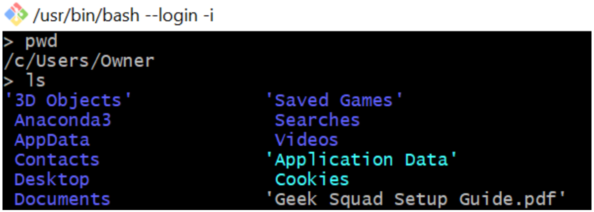


src:`http://korflab.ucdavis.edu/Unix_and_Perl/unix_and_perl_v3.0.html`

Navigating Your File System

The bash shell (Windows or Mac) will indicate that it is ready to receive your commands by prompting you with a **$**

- To identify where your shell is looking in your file tree, you can use the `pwd` command (print working directory) to print your location. Try it yourself (type the command in, and press enter)
- Use the `ls` command to 'list' all the files in your current directory (this may include other directories)



/usr/bin/bash --login -i

```
> pwd
/c/Users/Owner
> ls
'3D Objects'                'Saved Games'
 Anaconda3                   Searches
 AppData                     Videos
 Contacts                   'Application Data'
 Desktop                     Cookies
 Documents                  'Geek Squad Setup Guide.pdf'
```

Navigating Your File System

Suppose we want to move further down our file tree:

- We just used `ls` to identify subdirectories (folders) within our current directory.
- To move into one of these subfolders, type:
  `cd <dir name>` (which directory you choose is up to you)
  This command stands for 'change directory'
- You have now moved into the subdirectory, and can view the contents of this new folder with the `ls` command (see for yourself!)

Some Other Useful Navigation Tricks

Question: What if we want to go up the file tree instead of down?

Answer: Bash uses the argument `../` to mean 'the parent directory of my current location'

- So by typing:
  ```
  cd ../
  ```
  I move up one level of the tree.

- Note that `cd` also accepts longer file paths so by typing:
  ```
  cd ../../../
  ```
  I would move up three levels (assuming you are at least 3 levels down)

Some Other Useful Navigation Tricks

After a few commands in a row, you'll probably be close to the bottom of the shell:

- Use the `clear` command to remove the history shown in the shell (Bash still maintains your current directory, which you can confirm using `pwd` or `ls`)
- When moving down several directories at once, you can use the `TAB` key to autocomplete directory names as you type (this assumes the name is unique):
  ```
  cd Documents/gitrepos/JSC270.github.io/
  ```

Some Other Useful Navigation Tricks

Important: Because Git and Bash are Unix-based systems, file paths must use / forward slashes to separate directory names.

- You can use the ↑ and ↓ arrow keys to recall previous commands in the shell (even if you've used `clear` to remove them from view)
- At any time, you can close the shell by typing: `exit`
- When you do this, the Bash shell resets to the root directory, and you lose your command history

Tip: Bash is case-sensitive, and does not like spaces in file names. For easier navigation, you might try naming files using **snake case**, where all words are lowercase, and separated with underscores.

Question Time

Any questions about file navigation?

Creating, Removing, and Renaming Directories

We can do more than look at our file tree: we can **add new directories**

- We use the command `mkdir <newdir name>` to create a new directory (here, `<newdir name>` is something you choose)
- The command `rmdir <dir name>` will remove an existing directory
- You can use `mv <oldname> <newname>` to rename a directory. This command stands for 'move', since you can also use it to move a directory

Exercise 2: Creating a `gitrepos` Folder

It's standard practice to keep your local git repos in one place on your computer.

- **Let's create a `gitrepos` folder** (using the `mkdir` command) where we'll store your repo for assignment 2
- Where exactly you want to put this directory is up to you (I keep mine just inside my `Documents`).
- If you're not currently in the location you want, navigate through your file tree until you find your preferred location
- Please type **Done** into the Zoom chat once you've finished, so I know when to move forward

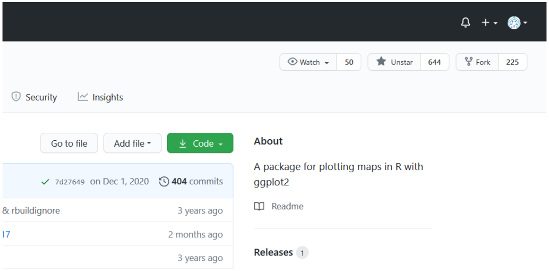
Forking or Cloning Existing Repos

- When you **clone** a remote repository, you're just making a copy of that repo on your local machine
- Anyone can clone any public GitHub repo
- However, if you're not a contributor on the repo you cloned:
  1. You cannot push local changes you make back to the remote (GitHub) repo
  2. You cannot pull new changes made by the contributors to update your local version (you would have to re-clone that remote repo if you wanted to update)
- The command to clone a repo is:
  `git clone http://github.com/username/repo-name`
  Where `username` and `repo-name` depend on the repo

Forking or Cloning Existing Repos

If you want to add to a GitHub repo but aren't the creator/contributor, you first need to **fork** that repo.

- **Forking** creates a copy of the repo you want on your account. Your copy is connected to the original repo, but changes you push to your forked copy do not affect the original



Forking or Cloning Existing Repos

- You can fork a repo using the 'fork' button in the upper right corner on the GitHub repo page (see picture above)
- Then you would clone your forked copy to your local machine (as before)
- To actually change the original repo, you would:
  1. Make changes locally, and push them to your forked copy (usually on a separate branch)
  2. Submit a **pull request** to the original repo owner, so they can merge the changes on your fork into the original remote repo

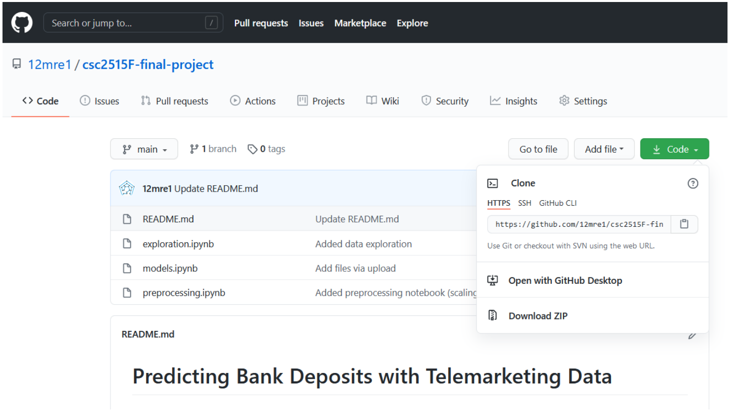
Forking or Cloning Existing Repos

- We won't be covering pull requests today
- They won't be neccessary for the assignment, since you won't need to modify Lauren's original repo (you're just pushing changes to your forked copy, which Lauren and I, as contributors, will see)
- However, if you would like to (one day) contribute to a large open-source project, you will likely use pull requests

Exercise 3: Cloning Your Repo

You'll need to fork a git repository for Assignment 2. Now, you can clone this repo to your local machine:

1. You can copy the link to your remote repository using the green button on GitHub



12mre1 Update README.md

README.md	Update README.md
exploration.ipynb	Added data exploration
models.ipynb	Add files via upload
preprocessing.ipynb	Added preprocessing notebook (scaling

```
https://github.com/12mre1/csc2515F-fin
```

Use Git or checkout with SVN using the web URL.

README.md

Predicting Bank Deposits with Telemarketing Data

Exercise 3: Cloning Your Repo

2. Make sure you're actually in your `gitrepos` directory (use the `cd` command if you're in its parent directory)
3. Clone your remote repo to your local machine by typing:
   `git clone <repo weblink>`

- Please type **Done** into the Zoom chat once you've finished, so I know when to move forward

Creating Files

In addition to creating directories, we can also create files:

- Use the command
  `touch <filename>`
  to initialize a file (e.g. `touch example.txt`)

- Then use
  `start <filename>`
  to open and edit that file

- We can use a similar syntax to open an application without specifying a file name:
  e.g. `start firefox`

- To remove a file, use: `rm <filename>`

Question Time

Any Questions?

Exercise 4: Adding a Text File to Your Assignment 2 Repo

Using the `touch` and `start` commands, create a file in your assignment 2 repo called `prereq.txt`

- In this text file, please indicate all the different types of statistical distributions you're familiar with (ie the ones you've worked with before)
- You can separate them with commas, or put one distribution on each line (the exact formatting doesn't matter)
- The next page contains a list of distributions for you to reference

Exercise 4: Adding a Text File to Your Assignment 2 Repo

Some types of distributions:

- Uniform
- Normal
- Bernoulli
- Binomial
- Exponential
- Geometric
- Hypergeometric
- Poisson
- Chi-squared
- Student-T
- F Distribution
- Gamma
- Beta
- Multinomial
- Negative Binomial
- Cauchy
- Laplace
- Pareto
- Rayleigh
- Weibull

When you've finished filling your text file, please save it and type **Done** in the Zoom Chat, so I know when to move forward.

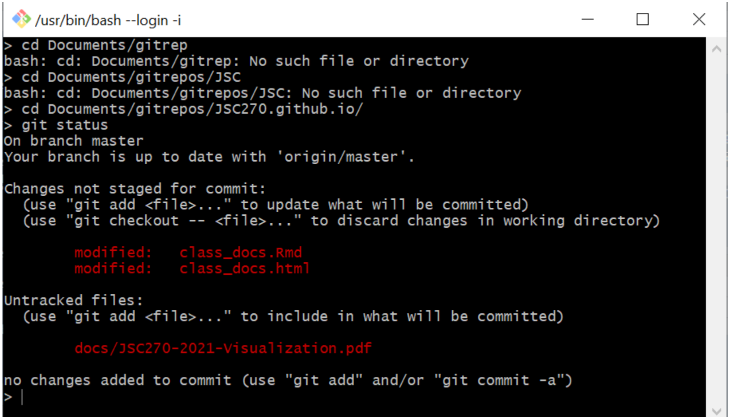
Tracking and Committing Changes

Once you've modified existing files or created new ones in your local git repo, we can use the process from last week to commit those changes:

1. Make sure you're in your git repo (use the `cd` command if not)
2. Use `git status` to check for modifications to the local repo
3. Use `git add <filename>` or `git add .` to stage your revisions
4. Use `git commit -m 'Message'` to commit the changes (take a snapshot)
5. Push the newest commit to the remote repo with git push

Exercise 5: Committing Your New File

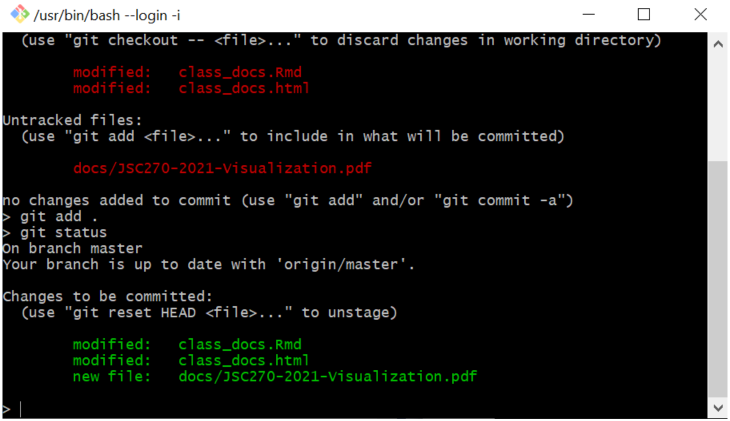
Go ahead and follow the steps above to add, commit, and push `prereq.txt` to your remote repository for Assignment 2



/usr/bin/bash --login -i

```
> cd Documents/gitrep
bash: cd: Documents/gitrep: No such file or directory
> cd Documents/gitrepos/JSC
bash: cd: Documents/gitrepos/JSC: No such file or directory
> cd Documents/gitrepos/JSC270.github.io/
> git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   class_docs.Rmd
        modified:   class_docs.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        docs/JSC270-2021-Visualization.pdf

no changes added to commit (use "git add" and/or "git commit -a")
> |
```

Exercise 5: Committing Your New File



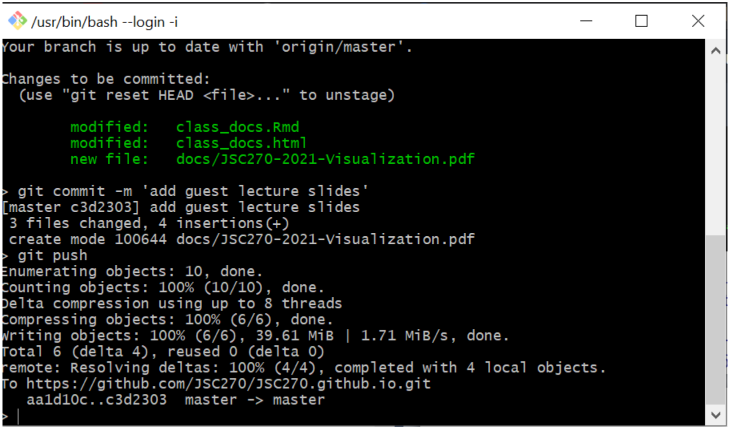
/usr/bin/bash --login -i

```
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   class_docs.Rmd
        modified:   class_docs.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        docs/JSC270-2021-Visualization.pdf

no changes added to commit (use "git add" and/or "git commit -a")
> git add .
> git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   class_docs.Rmd
        modified:   class_docs.html
        new file:   docs/JSC270-2021-Visualization.pdf

> |
```

You'll notice Bash uses different colours to separate modifications that haven't been staged (red) from those that are staged but not committed (green)

Exercise 5: Committing Your New File

You'll know you've succeeded when you get a message similar to:



```
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   class_docs.Rmd
        modified:   class_docs.html
        new file:   docs/JSC270-2021-Visualization.pdf

> git commit -m 'add guest lecture slides'
[master c3d2303] add guest lecture slides
 3 files changed, 4 insertions(+)
 create mode 100644 docs/JSC270-2021-Visualization.pdf
> git push
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 39.61 MiB | 1.71 MiB/s, done.
Total 6 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To https://github.com/JSC270/JSC270.github.io.git
   aa1d10c..c3d2303  master -> master
> |
```

Please type **Done** into the Zoom chat once you've successfully pushed changes.

Conclusion

In terms of Bash's capabilities, this is just the tip of the iceberg. You can:

- Customize the look of the shell
- Run programming files directly from the command line
- **Alias** certain commands (ie customize the shell commands to your liking)

But, we've covered everything you'll need to complete Assignment 2. Questions?