***5/21/2018 Notes:***

**GIT & GITHUB**

Git and github are 2 technologies that perform one function

***GIT*** = source control management(SCM) / backs up to cloud

Push code to repository (cloud) at least 1x per day

***REPO*** = repository

Keep repositories organized

Store all REPOs in c:\repos

DO NOT DO A GIT INIT IN REPOS!!!

~ = sitting in your home directory

Create all of our code inside c:\repos with subfolders



!!!NOTE that /c/repos is not a repository!!! --- The real repositories will be in subfolders

***SOURCE CONTROL*** = manages multiple versions of documents and keeps it in order

***GITHUB*** = same thing as GIT but in the cloud

Topics for GIT AND GITHUB today include: (1) Bash shell (see image above); (2) Command line interface (CLI); (3) File structure; and (4) Commands

***Bash shell:***

Similar to Linux command line

Commands are in book (pg.14 or so in book)

Also Greg will be getting us the slides for this presentation

***Command Line Interface (CLI):***

Git commands entered at command line (git cmd [opts}

In book on page 14

git: program name

cmd: what to do

[opts]: options based on command

All commands are lower case

>Git status –s

***File structure:***

How to tell if it is a file repository

Add –a to see hidden files

If it has a .git folder inside then it is a repository

You don’t want repositories inside of other repositories

Removing the .git folder removes git control

***Commands:***

All commands begin with git (lower case)

See GIT/GITHUB slide printouts for other commands such as init, remote, etc.

If you forget –m on message commits then add comments to top and “:X” in place of insert on text editor that pops up

Steps to add/commit things on github:(slide 17)

1. git init adds folder to start with git
2. git add to start tracking
3. git commit to commit (don’t forget to use -m for message)

***5/22/2018 Notes:***

***4 Announcements:***

1. Don’t leave electronics behind or other things behind each night
2. .NET meetup tonight @ 6PM
3. Party tomorrow @ 4:30PM
4. Fill out Day 1 feedback form

***SQL Servers:***

* ***SSMS*** = SQL Server Management Studio (interacts with the SQL table to manipulate it)
* Primary key is a unique identifier (at least one column but can be more than one) when taken together makes the value of that row unique – can use integers and for the bootcamp we will always use “Id” incremented 1 integer at a time (i.e., 1,2,3,4…)
* In SQL strings are bound by single quotes
* To insert a single quote (‘) in a string you must insert two single quotes next to each other (‘’); note that this is not a quotation mark but two single quotes
* Ordering can be different in SQL if your table is in different languages
* can't have any columns in the list that do not have a correspoding "group by" when using a sum
* when grouping data you must either group by or be aggregated or errors will be returned
* "having" works just like where but is used on aggregated data

***GitHub Upload of SQL Lesson:***

|  |
| --- |
| Max-Student@Student03 MINGW64 /c/repos  $ cd sqllesson  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git status  On branch master  Your branch is up to date with 'origin/master'.  Untracked files:  (use "git add <file>..." to include in what will be committed)  SqlLesson Queries/  nothing added to commit but untracked files present (use "git add" to track)  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote -v  origin https://github.com/gpdoud/SqlLesson.git (fetch)  origin https://github.com/gpdoud/SqlLesson.git (push)  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote remove origin  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote -v  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote add origin https://github.com/michaelbuchanan23/sqllesson.git  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git remote -v  origin https://github.com/michaelbuchanan23/sqllesson.git (fetch)  origin https://github.com/michaelbuchanan23/sqllesson.git (push)  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git status  On branch master  Untracked files:  (use "git add <file>..." to include in what will be committed)  SqlLesson Queries/  nothing added to commit but untracked files present (use "git add" to track)  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git add .  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git status  On branch master  Changes to be committed:  (use "git reset HEAD <file>..." to unstage)  new file: SqlLesson Queries/Assignment(TotalByState).sql  new file: SqlLesson Queries/Average.sql  new file: SqlLesson Queries/FromJupiterToSun.sql  new file: SqlLesson Queries/JoinSelectingOnlySpecificColumns.sql  new file: SqlLesson Queries/JoinViews.sql  new file: SqlLesson Queries/Name-City-State.sql  new file: SqlLesson Queries/OuterJoinWithNullValue.sql  new file: SqlLesson Queries/SelectAll.sql  new file: SqlLesson Queries/TotalSalesByCustomer.sql  new file: SqlLesson Queries/Union.sql  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git commit -m "Updated Sql Lesson with Queries from class on 5/22/2018"  [master 41f7d08] Updated Sql Lesson with Queries from class on 5/22/2018  10 files changed, 73 insertions(+)  create mode 100644 SqlLesson Queries/Assignment(TotalByState).sql  create mode 100644 SqlLesson Queries/Average.sql  create mode 100644 SqlLesson Queries/FromJupiterToSun.sql  create mode 100644 SqlLesson Queries/JoinSelectingOnlySpecificColumns.sql  create mode 100644 SqlLesson Queries/JoinViews.sql  create mode 100644 SqlLesson Queries/Name-City-State.sql  create mode 100644 SqlLesson Queries/OuterJoinWithNullValue.sql  create mode 100644 SqlLesson Queries/SelectAll.sql  create mode 100644 SqlLesson Queries/TotalSalesByCustomer.sql  create mode 100644 SqlLesson Queries/Union.sql  Max-Student@Student03 MINGW64 /c/repos/sqllesson (master)  $ git push origin master  Counting objects: 43, done.  Delta compression using up to 4 threads.  Compressing objects: 100% (39/39), done.  Writing objects: 100% (43/43), 9.06 KiB | 927.00 KiB/s, done.  Total 43 (delta 6), reused 0 (delta 0)  remote: Resolving deltas: 100% (6/6), done.  To https://github.com/michaelbuchanan23/sqllesson.git  \* [new branch] master -> master |

***5/23/2018:***

* SQL server 2017 google search will yield lots of results on the topic as will searches for C#
* Coding from scratch helps ingrain things better than copy and paste so code from scratch where possible
* Create apps in 2 steps:

1. Make it work/***FUNCTIONAL*** no matter how ugly it looks; and
2. Make it ***ELEGANT*** meaning:
   1. Robust meaning it doesn’t break;
   2. It is easily enhanced by someone else; and
   3. It’s efficient

* How do we keep learning:
  + Meetups
  + CLEs
  + Pro-bono work
* Make sure to verify your understanding of what an individual wants in their program
  + Verify by reflection – repeat back the request in your own words
* Join customer and order, sort by order amount biggest first so desc, no id fields but all other columns in
* Functions page for strings on Microsoft website:
* 
* When doing a delete make sure there is a where clause or it will delete everything…use where clause even if you intend to delete everything
* You can highlighted certain portions of code in Sql Server if you only want to run that particular portion of the code
* Comment group of lines shortcut key is ctrl+k+c
* Uncomment group of lines shortcut key is ctrl+k+u
* Read the data you want to change first via UPDATE with a where clause then copy that WHERE clause to your update in order to accurately update the data
* Don’t ever update your primary key
* The only time (usually) you should use null with numbers is when using foreign keys
* nvarchar is now the new way to define a string because of international languages where you need a double byte
* zip codes should always be strings because it doesn’t represent a mathematical number

***Notes 5/24/2018:***

* *Interviews*: Be succinct during interviews in your answer unless you are asked to elaborate

***SQL Server:***

* When changing the table definition the syntax is ALTER TABLE [name of table]
  + Usually used to add/drop column or change the type (e.g., change zip code from int to string)
* Purchase Request System (PRS) Tables:
  + User: list of users authorized to log in. A user is the owner of a purchase request.
  + Vendor: company that sells products
  + Product: an item that can be requested by a user on a purchase request
  + Purchase Request: grouping of all items a user requests (i.e., instance of a request containing user and total)
  + Purchase Request Line Item: a single product and quantity on a purchase request
* To create a unique inex on a table:
  + Create (unique) index [name] on [tablename](col1,col2,…)
* Stored Procedures:
  + Basically creating functions for sql
  + You may also just do queries in stored procedures (in cases like where variables are needed like different users production per week which will change with each query meaning users and the week will change with each query)
  + Stored procedures vary wildly across different database systems (i.e., MySQL vs. SQL Server)
  + Code block is the same syntactically as one line of code – between begin and end of stored procedure is a code block
* Create stored procedure to load data into tables for each of our new tables called “Add” and “Table Name”

***Notes 5-25-2018:***

***TO DO: COMBINE NOTES INTO ONE DOCUMENT***

***GIT/GITHUB Assignment Information:***

* We will push our assignments to greg via <https://github.com/gpdoud/dotnet4-assignments.git>
* “git pull origin master” is how we pull
* If we are having issues with pushing then we should pull down the most recent version of the folder via “git pull origin master” then attempt to “git push origin master” again

***Code blocks:***

* Is either a single statement or
* Begin…End

***T-SQL:***

* Allows one to code procedures in sql server

***General Coding Note:***

* If you are nesting your if statements more than 3 levels deep then you need to reevaluate how you’ve done it and do it more efficiently and easier to read
* When you need to know whether a certain kind of data exists then you use an Exist – 1 statement such as in “Select1-LookFor1InstanceAndReturnTwoDifferingSentences(UseForTransactions).sql”
* Looping does a group of statements potentially more than once until some Boolean condition is true or false
* Default values in SQL are all null