# CS601: Software Development for Scientific Computing

Autumn 2021

Week3: Structured Grids (Contd..), Version Control System (Git and GitHub), Intermediate C++

# Last Week..

- Program Development Environment – Demo
- ‘C’ subset of C++ and reference variables in C++
- Discretization and issues
  - scalability, approximation, and errors (discretization error and solution error), error estimates
  - mesh of cells/elements, cell shapes and sizes
- Structured Grids
  - ‘Regularity’ of cell connectivity (e.g. neighbors are similar kind of cells)
  - Case study – problem statement, representation (e.g. 2D arrays)

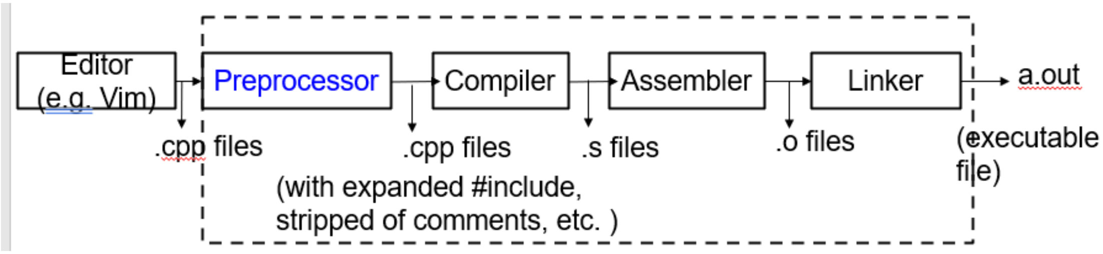
# Review of Solution to Exercise: Product of Vectors

- Input sanity check using `istringstream`
- Good programming style: separation of the interface from implementation
  - Streams
  - Passing arrays to functions
  - Pragmas and preprocessor directives
  - Namespaces
- In the sample code, we have so many versions!

*Demo*

# Detour - Conditional Compilation

- Set of 6 preprocessor directives and an operator.
  - #if
  - #ifdef
  - #ifndef
  - #elif
  - #else
  - #endif
- Operator ‘defined’



# #if

```
#if <constant-expression>
cout<<“CS601”;
#endif
```

←── //This line is compiled only if `<constant-expression>` evaluates to a value > 0 while preprocessing

```
#define COMP 0
#if COMP
cout<<“CS601”
#endif
```

*No compiler error*

```
#define COMP 2
#if COMP
cout<<“CS601”
#endif
```

*Compiler throws error about missing semicolon*

# #ifdef

```
#ifdef identifier
cout<<"CS601";
#endif
```

//This line is compiled only if identifier is defined before the previous line is seen while preprocessing.

identifier does not require a value to be set. Even if set, does not care about 0 or > 0.

```
#define COMP
#ifdef COMP
cout<<"CS601"
#endif
```

```
#define COMP 0
#ifdef COMP
cout<<"CS601"
#endif
```

```
#define COMP 2
#ifdef COMP
cout<<"CS601"
#endif
```

*All three snippets throw compiler error about missing semicolon*

# #else and #elif

```
1. #ifdef identifier1
2. cout<<“Summer”
3. #elif identifier2
4. cout<<“Fall”;
5. #else
6. cout<<“Spring”;
7. #endif
```

//preprocessor checks if identifier1 is defined. if so, line 2 is compiled. If not, checks if identifier2 is defined. If identifier2 is defined, line 4 is compiled. Otherwise, line 6 is compiled.

# `defined` operator

**Example:**

```
#if defined(COMP)
cout<<"Spring";
#endif
```

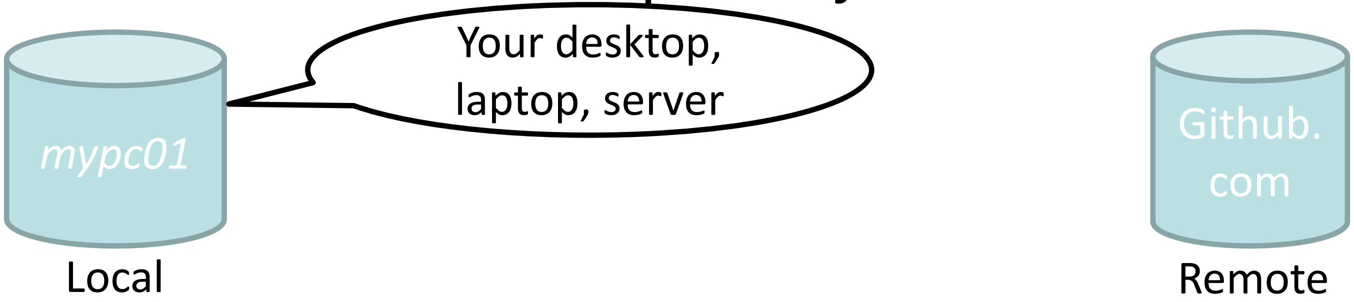
//same as if #ifdef COMP

```
#if defined(COMP1) || defined(COMP2)
cout<<"Spring";
#endif
```

//if either `COMP1` or `COMP2` is defined, the `printf` statement is compiled. As with `#ifdef,` `COMP1` or `COMP2` values are irrelevant.

# Git

- Example of a Version Control System
  - Manage versions of your code – access to different versions when needed
  - Lets you collaborate
- ‘Repository’ – term used to represent storage
  - *Local* and *Remote* Repository



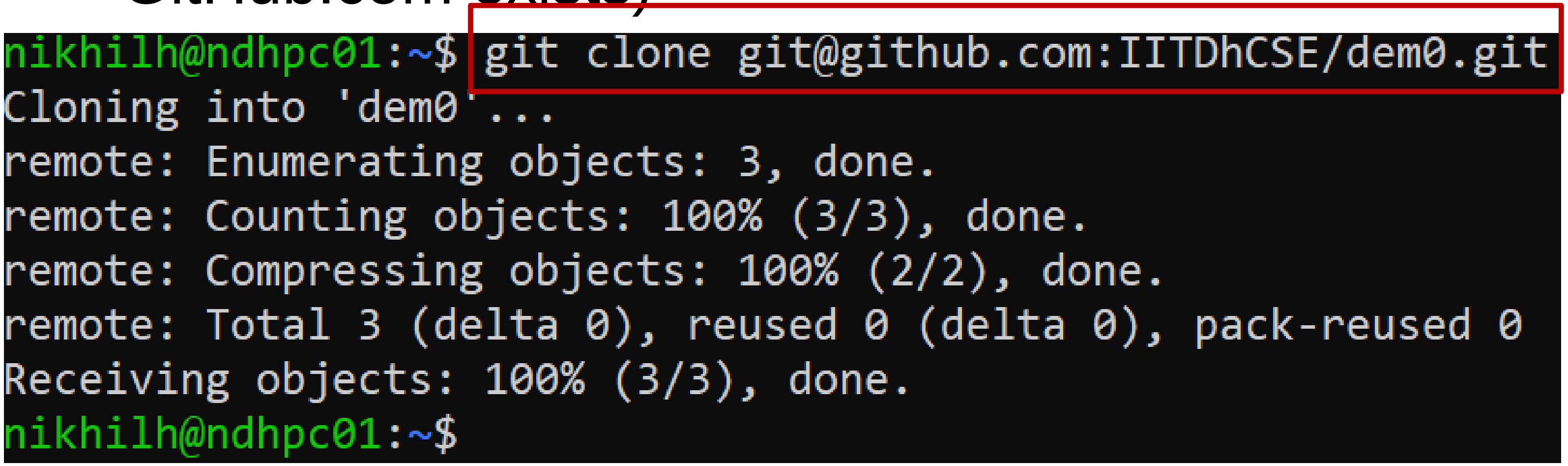
# Git – Creating Repositories

- Two methods:
  1. ‘Clone’ / Download an existing repository from GitHub
  2. Create local repository first and then make it available on GitHub

# Method 1: `git clone` for creating local working copy

– ‘Clone’ / Download an existing repository from GitHub – get your own copy of source code

- `git clone` (when a remote repository on GitHub.com exists)



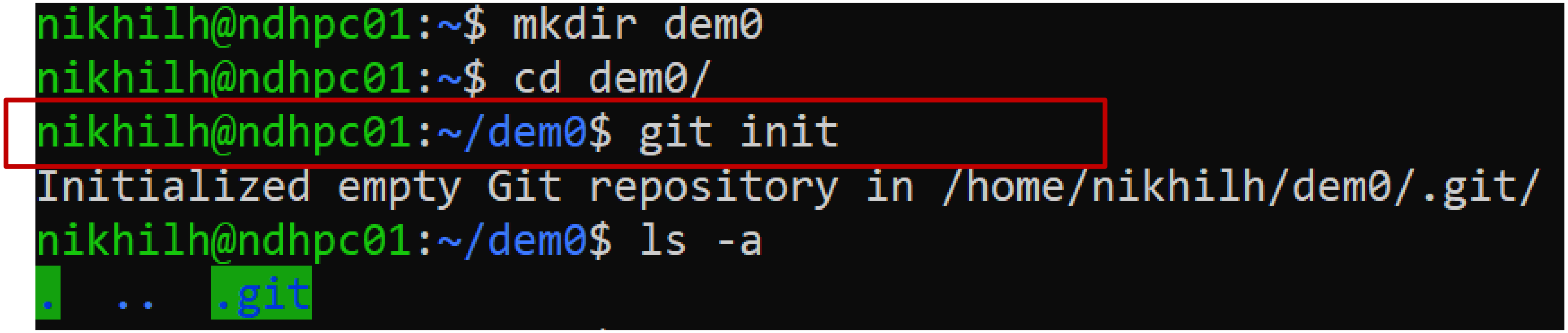
```
nikhilh@ndhpc01:~$ git clone git@github.com:IITDhCSE/dem0.git
Cloning into 'dem0'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
nikhilh@ndhpc01:~$
```

# Method 2: `git init` for initializing local repository

– Create local repository first and then make it available on GitHub

1. `git init`

converts a directory to Git local repo

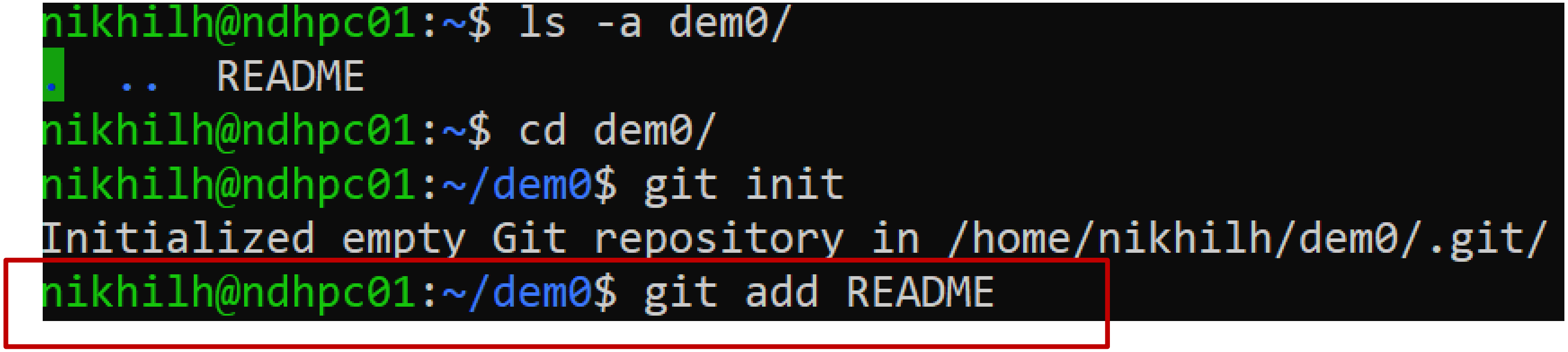


```
nikhilh@ndhpc01:~$ mkdir dem0
nikhilh@ndhpc01:~$ cd dem0/
nikhilh@ndhpc01:~/dem0$ git init
Initialized empty Git repository in /home/nikhilh/dem0/.git/
nikhilh@ndhpc01:~/dem0$ ls -a
.  ..  .git
```

# git add for staging files

2. git add

‘stage’ a file i.e. prepare for saving the file on local repository



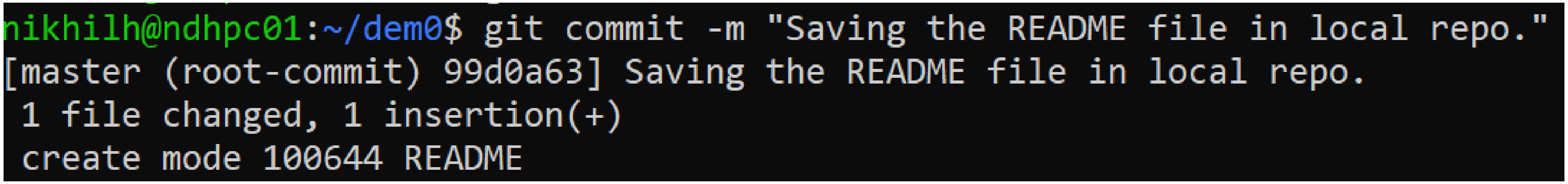
```
nikhilh@ndhpc01:~$ ls -a dem0/
.   ..   README
nikhilh@ndhpc01:~$ cd dem0/
nikhilh@ndhpc01:~/dem0$ git init
Initialized empty Git repository in /home/nikhilh/dem0/.git/
nikhilh@ndhpc01:~/dem0$ git add README
```

Note that creating a file, say, README2 in dem0 directory does not *automatically* make it part of the local repository

# `git commit` for saving changes in local repository

3. `git commit`

‘commit’ changes i.e. save all the changes (adding a new file in this example) in the local repository



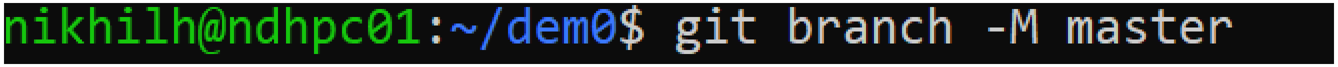
```
nikhilh@ndhpc01:~/dem0$ git commit -m "Saving the README file in local repo."
[master (root-commit) 99d0a63] Saving the README file in local repo.
 1 file changed, 1 insertion(+)
 create mode 100644 README
```

*How to save changes done when you must overwrite an existing file?*

# Method 2 only: `git branch` for branch management

```
4. git branch –M master
```

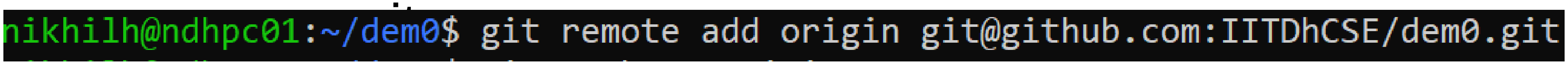
rename the current as ‘`master`’ (-M for force rename even if a branch by that name already exists)



```
nikhilh@ndhpc01:~/dem0$ git branch -M master
```

# Method 2 only: `git remote add`

5. `git remote add origin git@github.com:IITDhCSE/dem0.git` – prepare the local repository to be managed as a tracked



```
nikhilh@ndhpc01:~/dem0$ git remote add origin git@github.com:IITDhCSE/dem0.git
```

command to manage remote repo.

associates a name ‘origin’ with the remote repo’s URL

The URL of the repository on GitHub.com.

- This URL can be that of any other user’s or server’s address.
- uses SSH protocol
  - HTTP protocol is an alternative. Looks like: https://github.com/IITDhCSE/dem0.git

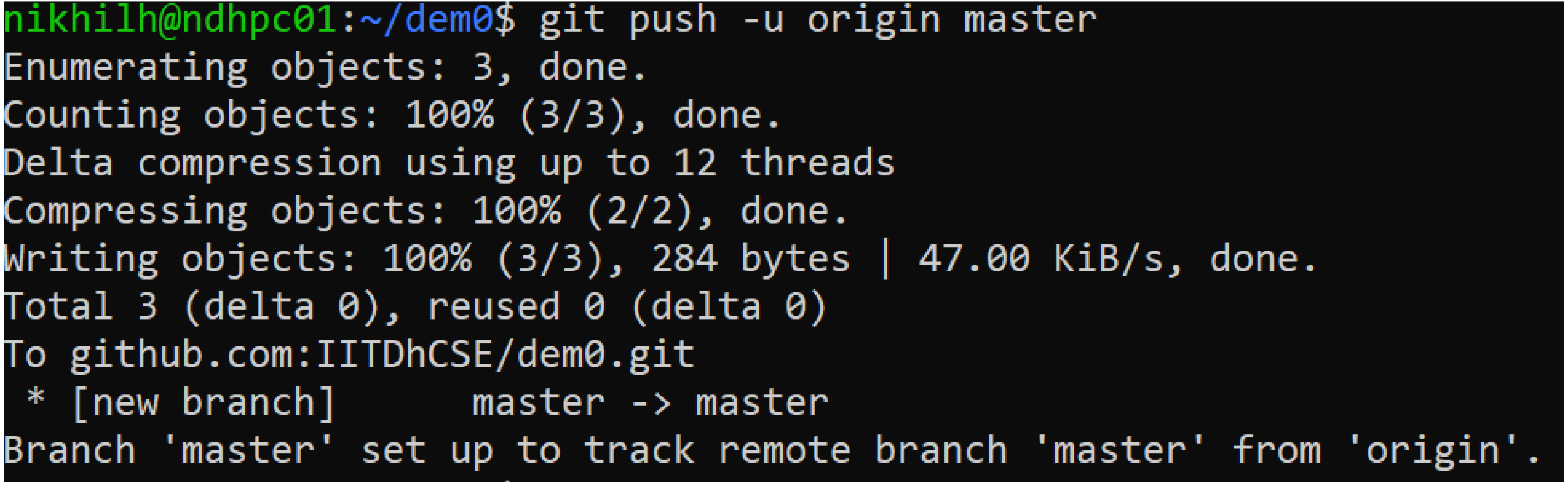
# Method 2 only: GitHub Repository Creation

5.a) Create an empty repository on GitHub.com

(name must be same as the one mentioned previously – `dem0`)

# `git push` for saving changes in remote repo

6. `git push –u origin master` – 'push' or save all the changes done to the 'master' branch in local repo to remote repo. *(necessary for guarding against deletes to local repository)*



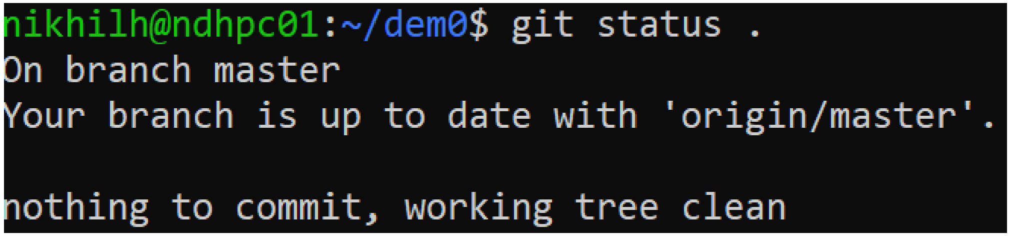
```
nikhilh@ndhpc01:~/dem0$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 284 bytes | 47.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:IITDhCSE/dem0.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

`syntax: git push <remotename> <branchname>`

*what does the –u option do?*

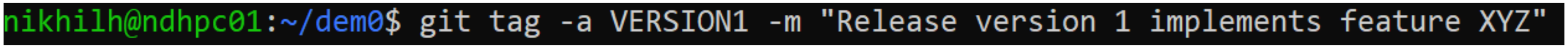
# Git – Releasing Code

- Tagging
  - Check for unsaved changes in local repository.



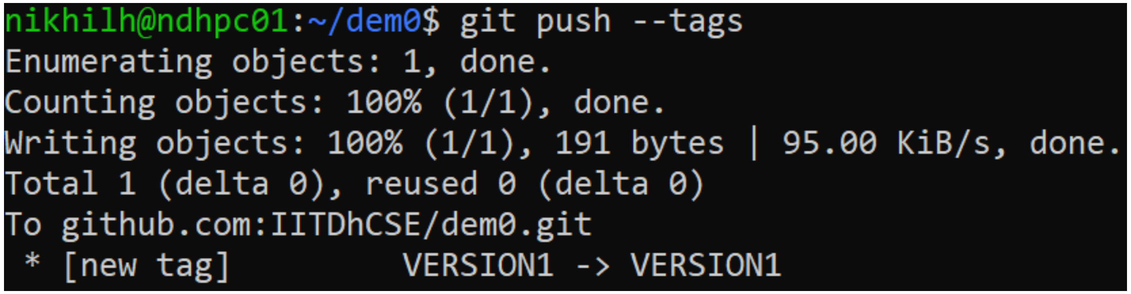
```
nikhilh@ndhpc01:~/dem0$ git status .
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

  - Create a tag and associate a comment with that tag



```
nikhilh@ndhpc01:~/dem0$ git tag -a VERSION1 -m "Release version 1 implements feature XYZ"
```

  - Save tags in remote repository



```
nikhilh@ndhpc01:~/dem0$ git push --tags
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 191 bytes | 95.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To github.com:IITDhCSE/dem0.git
 * [new tag]         VERSION1 -> VERSION1
```

# Git – Recap..

```
1.  git clone (creating a local working copy)
2.  git add (staging the modified local copy)
3.  git commit (saving local working copy)
4.  git push (saving to remote repository)
5.  git tag (Naming the release with a label)
6.  git push --tags (saving the label to remote)
```

- Note that commands 2, 3, and 4 are common to Method 1 and Method 2.
- Please read https://git-scm.com/book/en/v2 for details