CS35L Software Construction Laboratory

Lab 5: Sneha Shankar

Week 9; Lecture 2

Dynamic Loading

to let an application load and link libraries itself

- application can specify a particular library to load, then
- application can call functions within that library

load shared libraries from disk (file) into memory and re-adjust its location

done by a library named ld-linux.so.2

the Dynamic Loading API

dlopen - makes an object file accessible to a program

void *dlopen(const char *file, int mode);

RTLD NOW → relocate now; RTLD LAZY → to relocate when needed;

dlsym - gives resolved address to a symbol within this object

void *dlsym(void *restrict handle, const char *restrict name);

check char *dlerror(); if an error occurs

dlerror - returns a string error of the last error that occurred

dlclose - closes an object file

Worth a read:

https://www.dwheeler.com/program-library/Program-Library-HOWTO/x172.html

Dynamic loading

```
#include <stdio.h>
#include <dlfcn.h>

int main(int argc, char* argv[]) {
  int i = 10;
  void (*myfunc)(int *); void *dl_handle;
  char *error;
  dl_handle = dlopen("libmymath.so", RTLD_LAZY);//RTLD_NOW
  if(!dl_handle) {
    printf("dlopen() error - %s\n", dlerror()); return 1;
  }
  //Calling mul5(&i);
  myfunc = dlsym(dl_handle, "mul5"); error = dlerror();
  if(error != NULL) {
    printf("dlsym mul5 error - %s\n", error); return 1;
  }
  myfunc(&i);
  //Calling add1(&i);
  myfunc = dlsym(dl_handle, "add1"); error = dlerror();
  if(error != NULL) {
    printf("dlsym add1 error - %s\n", error); return 1;
  }
  myfunc(&i);
  printf("i = %d\n", i);
  dlclose(dl_handle);
  return 0;
}
```

Creating static and shared libs in GCC

- mymath.h

```
#ifndef _ MY_MATH_H

#define _ MY_MATH_H

void mul5(int *i);

void add1(int *i);

#endif
```

- mul5.c

```
#include "mymath.h"

void mul5(int *i)

{

  *i *= 5;

}
```

- add1.c

```
#include "mymath.h"

void add1(int *i)

{

  *i += 1;

}
```

- gcc -c mul5.c -o mul5.o
- gcc -c add1.c -o add1.o
- ar   -cvq libmymath**.a** mul5.o add1.o ----> (static lib)
- gcc -**shared** -fpic -o libmymath**.so** mul5.o add1.o -----> (shared lib)

Homework 8

the homework - to split an application into dynamically linked modules
randall.c = randcpuid.c + randlibhw.c + randlibsw.c + randmain.c

randall.c =
randcpuid.c + randlibhw.c + randlibsw.c + randmain.c

1. build the libraries
2. load the libraries
3. run the functions in libraries

Homework 8

Flags:

```
gcc -shared -fPIC greeting-fr.c -o greeting-fr.so
gcc -ldl -Wl,-rpath=. greeting-dl.c -o greet-dl
```

- -fPIC to output position independent code
- -lmylib to link with \libmylib.so"
- -L to nd .so les from this path, default is /usr/lib
- -Wl,rpath=dir to set rpath option to be dir to linker (by using -Wl)
- -shared to build a shared object

Attribute of functions:

__attribute__ ((constructor)) to run when dlopen() is called
__attribute__ ((destructor)) to run when dlclose() is called

Week 10: Change Management

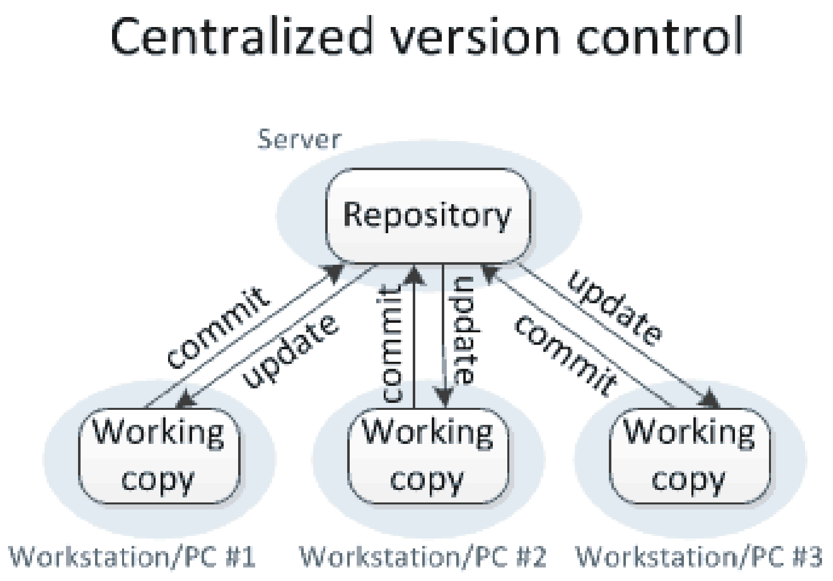
Software development process

- Involves making a lot of changes to code
  - New features added
  - Bugs fixed
  - Performance enhancements
- Software team has many people working on the same/different parts of code
- Many versions of software released
  - Ubuntu 10, Ubuntu 12, etc
  - Need to be able to fix bugs for Ubuntu 10 for customers using it, even though you have shipped Ubuntu 12.

Source/Version Control

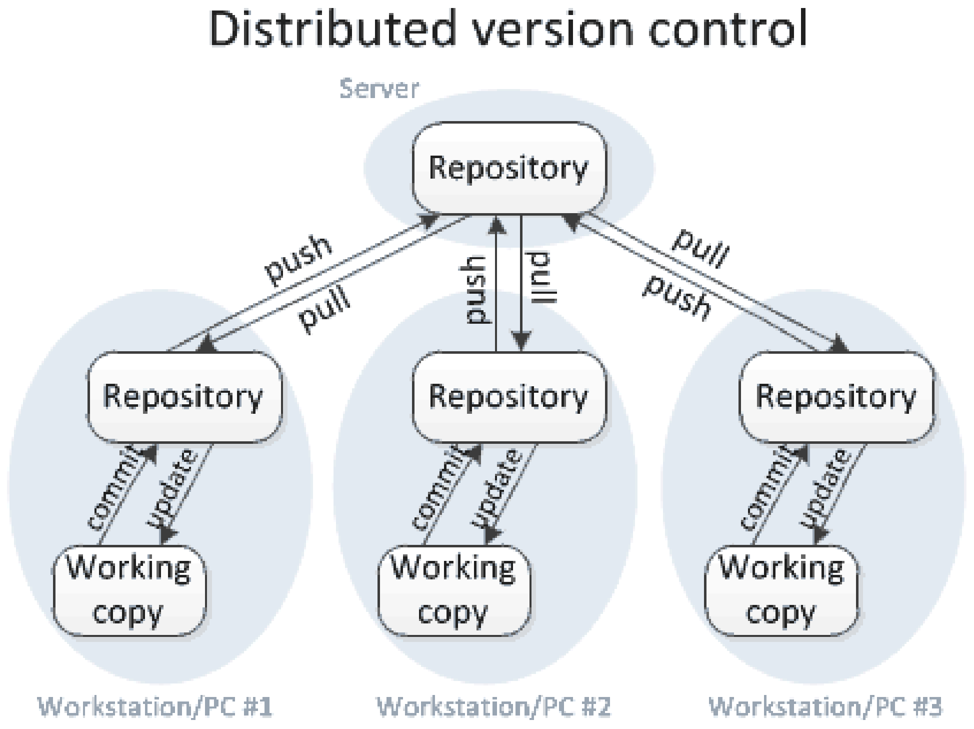
- Track changes to code and other files related to the software
  - What new files were added?
  - What changes made to files?
  - Which version had what changes?
  - Which user made the changes?
- Track entire history of the software
- Version control software
  - GIT, Subversion, Perforce

Centralized VCS



- Version history sits on a central server
- Users will get a working copy of the files
- Changes have to be committed to the server
- All users can get the changes

Distributed VCS



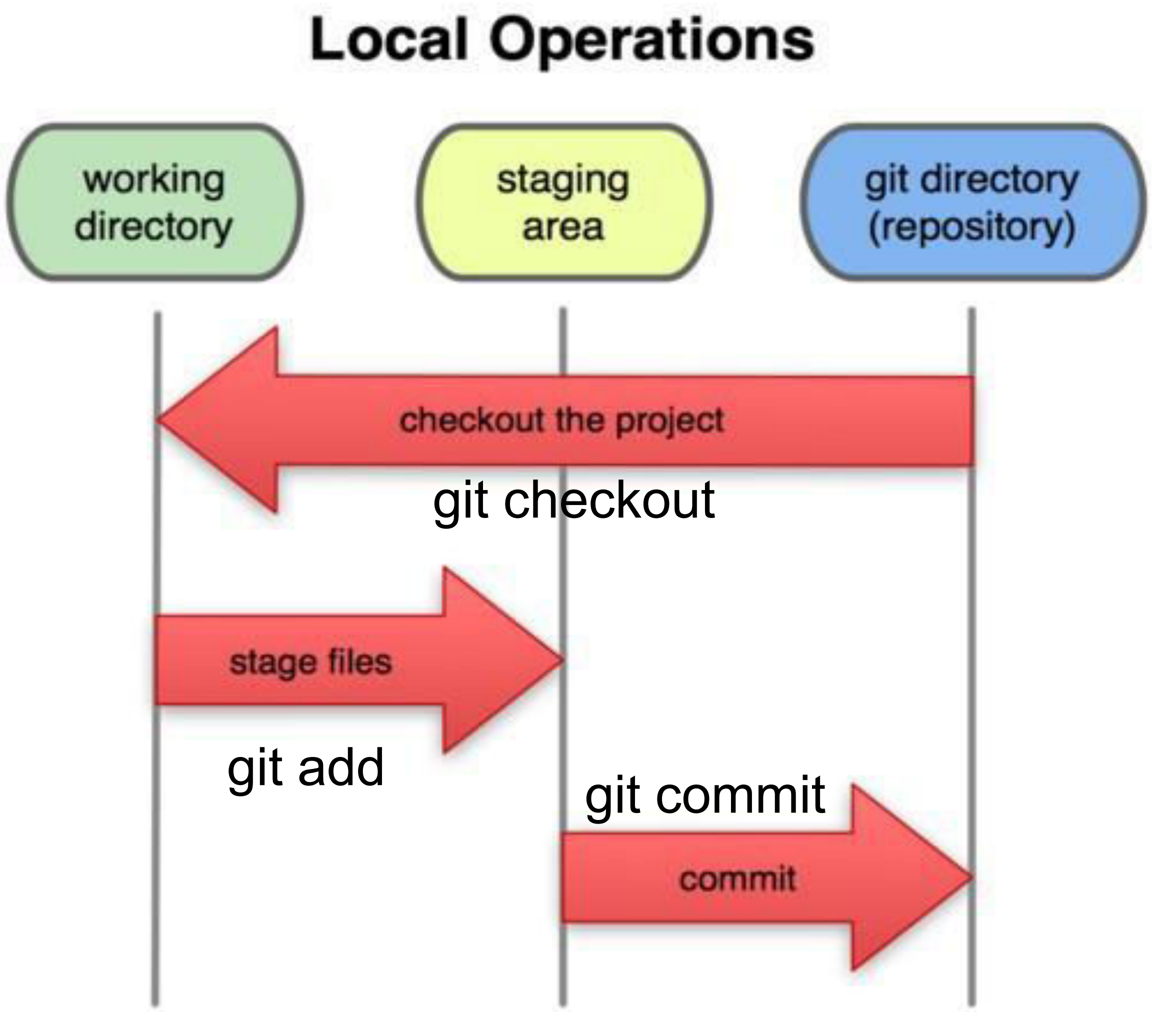
- Version history is replicated at every user's machine
- Users have version control all the time
- Changes can be communicated between users
- Git is distributed

Terms used

- **Repository**
  - Files and folder related to the software code
  - Full history of the software
- **Working copy**
  - Copy of software's files in the repository
- **Check-out**
  - To create a working copy of the repository
- **Check-in / Commit**
  - Write the changes made in the working copy to the repository
  - Commits are recorded by the VCS

GIT Source control

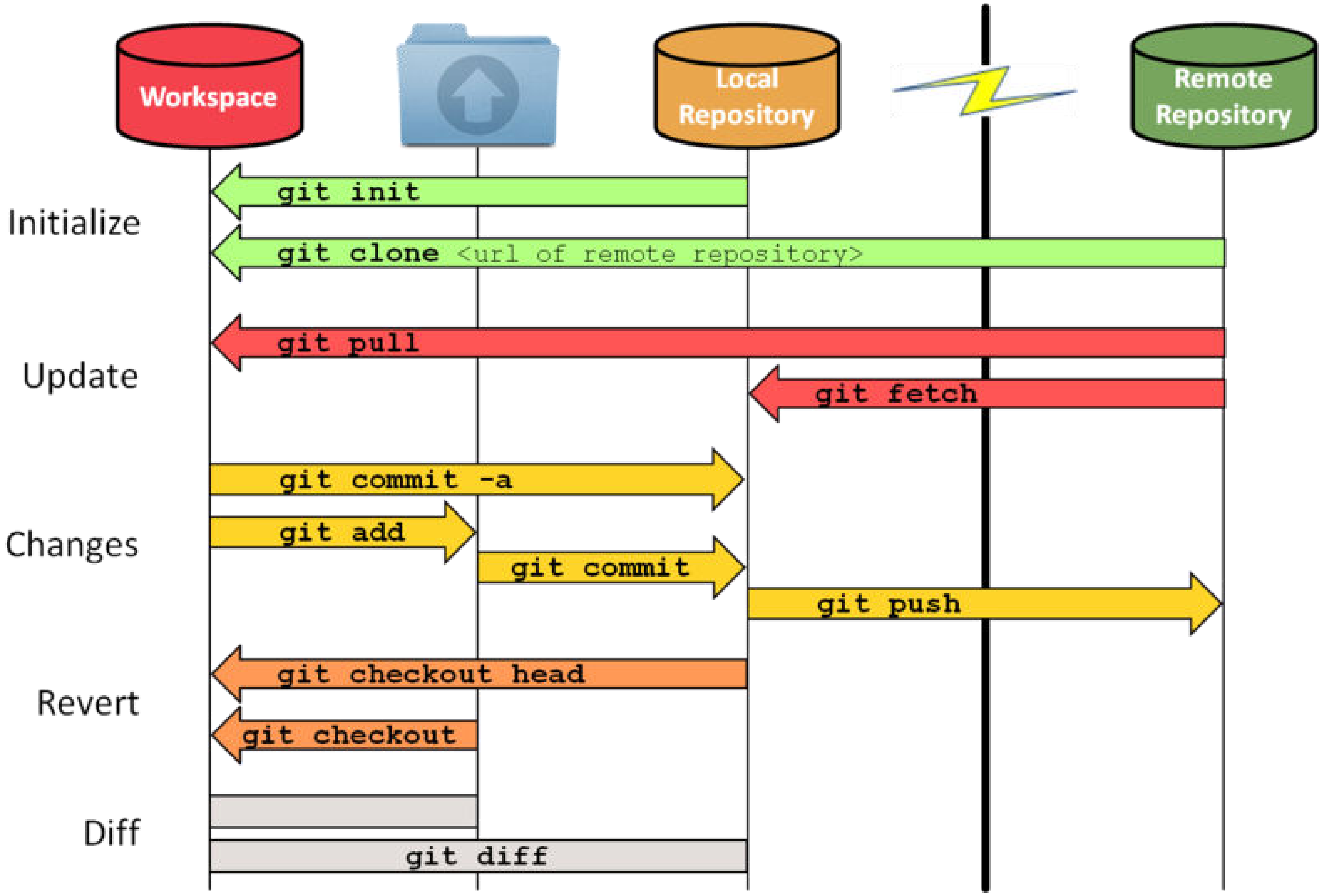
Git States

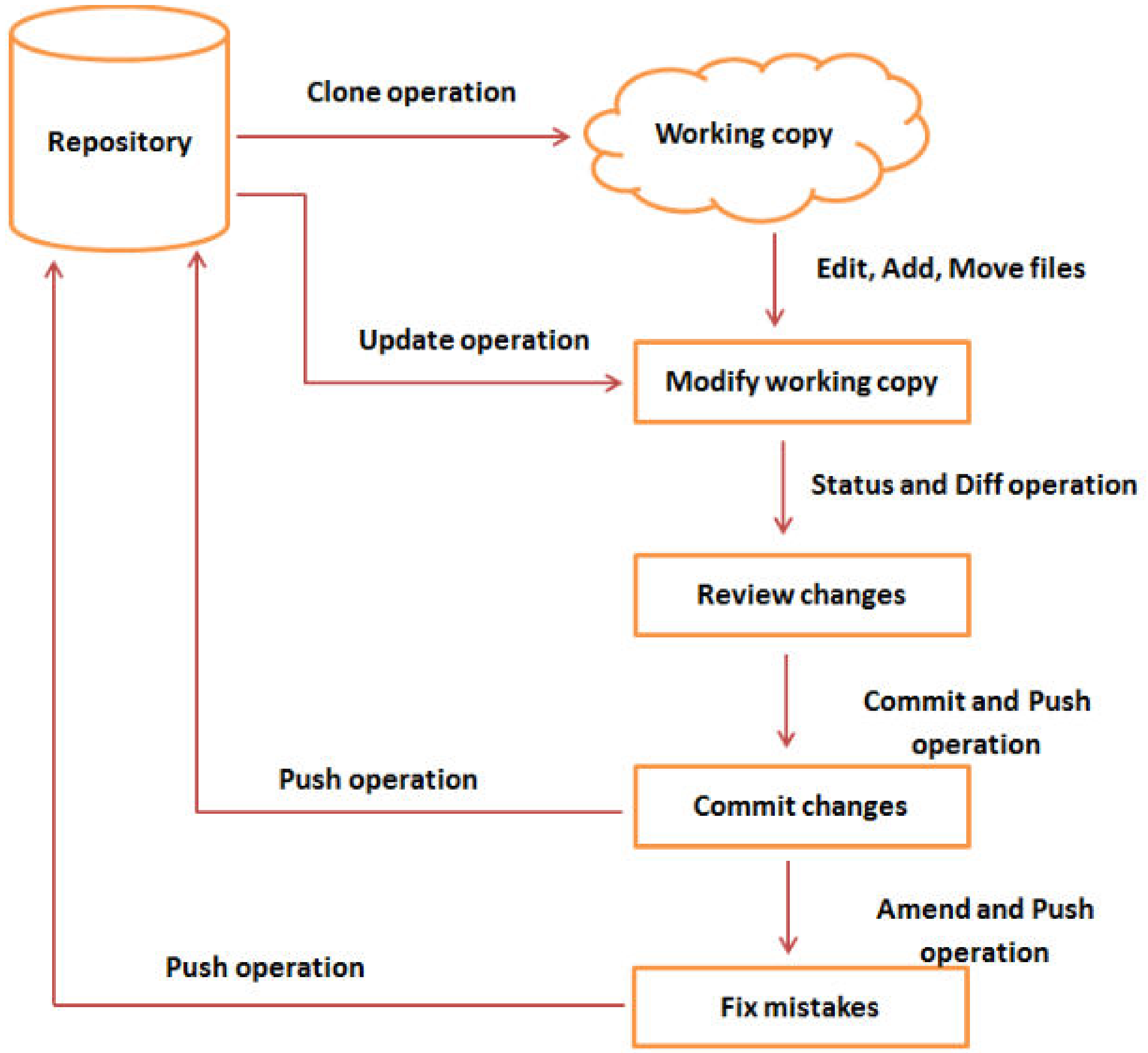


git checkout

git add

git commit


Workspace
Local Repository
Remote Repository
Initialize
git init
git clone <url of remote repository>
Update
git pull
git fetch
Changes
git commit -a
git add
git commit
git push
Revert
git checkout head
git checkout
Diff
git diff


Repository
Clone operation
Working copy
Edit, Add, Move files
Update operation
Modify working copy
Status and Diff operation
Review changes
Commit and Push operation
Push operation
Commit changes
Amend and Push operation
Push operation
Fix mistakes

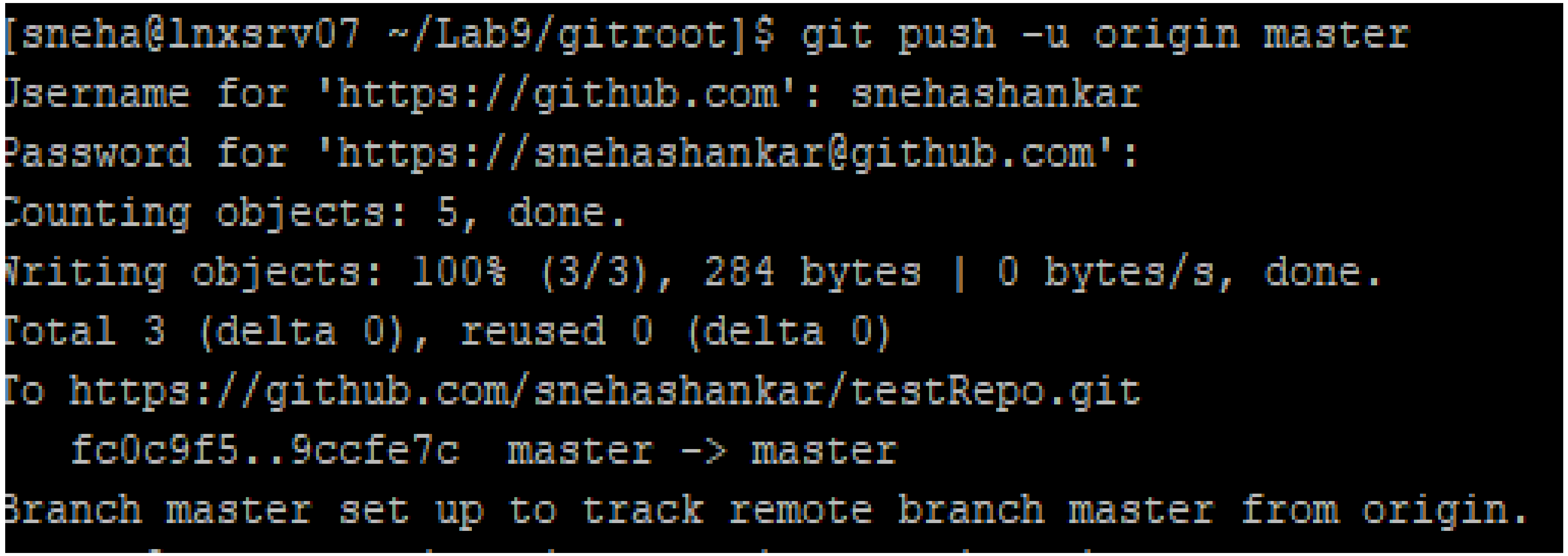
First Git Repository

- $ `mkdir gitroot`
- $ `cd gitroot`
- $ `git init`
  - creates an empty git repo (.git directory)

- $ `echo "Hello World" > hello.txt`
- $ `git add .`
  - Adds content to the index
  - Must be run prior to a commit
- $ `git commit –m 'Check in number one'`

Push changes to remote repository

- git remote add origin https://github.com/snehashankar/testRepo.git
- git push –u origin master



```
[sneha@lnxsrv07 ~/Lab9/gitroot]$ git push -u origin master
Username for 'https://github.com': snehashankar
Password for 'https://snehashankar@github.com':
Counting objects: 5, done.
Writing objects: 100% (3/3), 284 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/snehashankar/testRepo.git
   fc0c9f5..9ccfe7c  master -> master
Branch master set up to track remote branch master from origin.
```

Working With Git

- `$ echo “I love Git” >> hello.txt`
- `$ git status`
  - Shows list of modified files
  - hello.txt
- `$ git diff`
  - Shows changes we made compared to index
- `$ git add hello.txt`
- `$ git diff`
  - No changes shown as diff compares to the index
- `$ git diff HEAD`
  - Now we can see changes in working version
- `$ git commit –m “Second commit”`

Git commands

- Repository creation
  - $ git init (Start a new repository)
  - $ git clone (Create a copy of an exisiting repository)
- Branching
  - $ git checkout <tag/commit> -b <new_branch_name> (creates a new branch)
- Commits
  - $ git add (Stage modified/new files)
  - $ git commit (check-in the changes to the repository)
- Getting info
  - $ git status (Shows modified files, new files, etc)
  - $ git diff (compares working copy with staged files)
  - $ git log (Shows history of commits)
  - $ git show (Show a certain object in the repository)
- Getting help
  - $ git help