

Lab 1

COP 3402
(Aug 29, 2018)

Outline

- C Programming
 - How to build a makefile
- Git Tutorial
 - What is git?
 - How to use it?



The Makefile Utility



Motivation

- Small programs
 - Single file.
- “Not so small” programs:
 - Many lines of code.
 - Multiple components.
 - More than one programmer.

Large Projects

- Problems:
 - Long files are harder to manage (for both programmers and machines).
 - Every change requires long compilation.
 - Many programmers can not modify the same file simultaneously.

Large Projects

- How to handle large projects?
 - divide project to multiple files.
- Targets:
 - Good **division** to components.
 - **Minimum compilation** when something is changed.
 - **Easy maintenance** of project structure, dependencies and creation.

Compilation Process

- Converting written code to Machine code.
- Example:
 - main.c

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

```
int main() {  
    printf("hello world!\n");  
    return 0;  
}
```

Compilation Process

- In the directory there exists only
 - main.c
- gcc -c main.c (compile)
 - This will generate the main.o file from main.c
- gcc -o myprogram main.o (linking)
 - This will generate the executable from the .o file.
- ./myprogram

Compilation Process Example

- Example:
 - Simple Project:
 - main.c
 - Bigger Project:
 - main.c sum.c sum.h
 - gcc -c main.c
 - gcc -c sum.c
 - gcc -o sum main.o sum.o
 - ./sum

Project maintenance when we have multiple files

- A **makefile** is a file (script) containing:
 - Project **structure** (files, **dependencies**).
 - **Instructions** for files creation.
- The **make** command reads a makefile, understands the project structure and creates the executable.



Example:

- Program contains 3 files:
 - `main.c` `sum.c` `sum.h`
- `sum.h` included in `both` `.c` files.
- Executable should be the file `sum`.

makefile

```
sum: main.o sum.o  
    gcc -o sum main.o sum.o
```

Rule 1

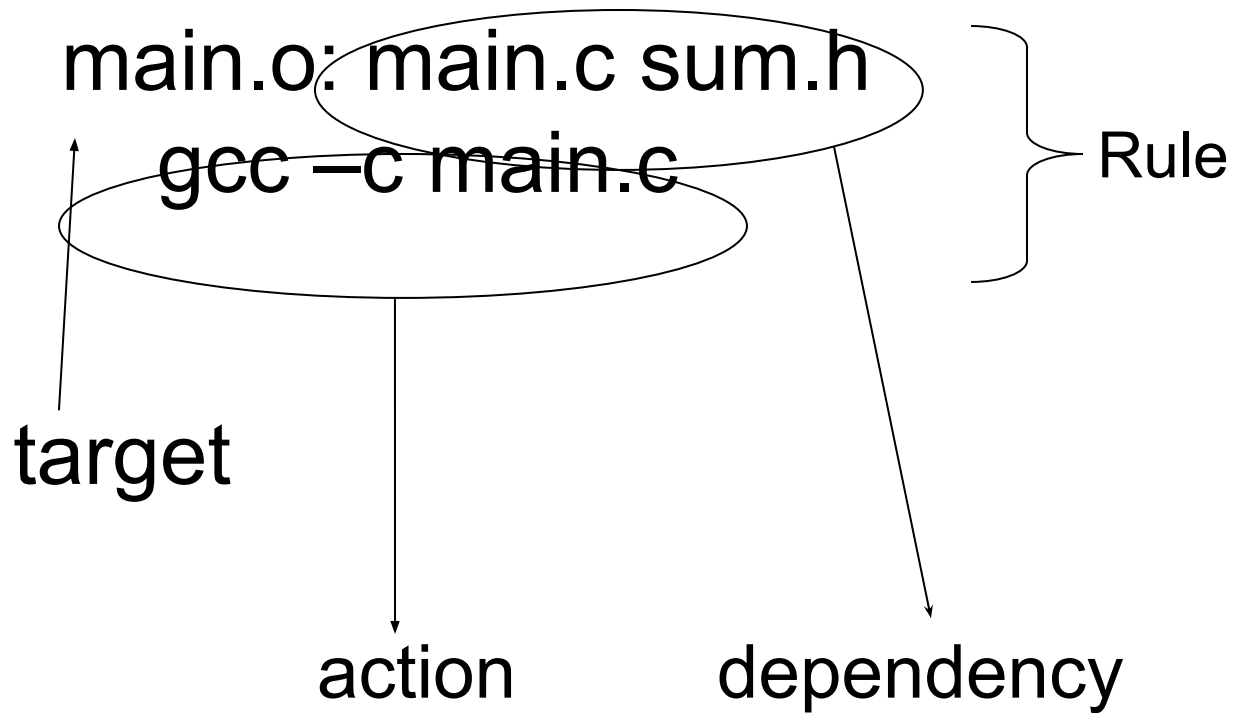
```
main.o: main.c sum.h  
    gcc -c main.c
```

Rule 2

```
sum.o: sum.c sum.h  
    gcc -c sum.c
```

Rule 3

Rule Syntax



makefile



➡ `sum: main.o sum.o`
➡ `gcc -o sum main.o sum.o`



➡ `main.o: main.c sum.h`
➡ `gcc -c main.c`



➡ `sum.o: sum.c sum.h`
➡ `gcc -c sum.c`

`main.c`
`sum.c`
`sum.h`
`main.o`
`sum.o`
`sum`

Equivalent makefiles

- .o depends (by default) on corresponding .c file.
Therefore, an equivalent makefile is:

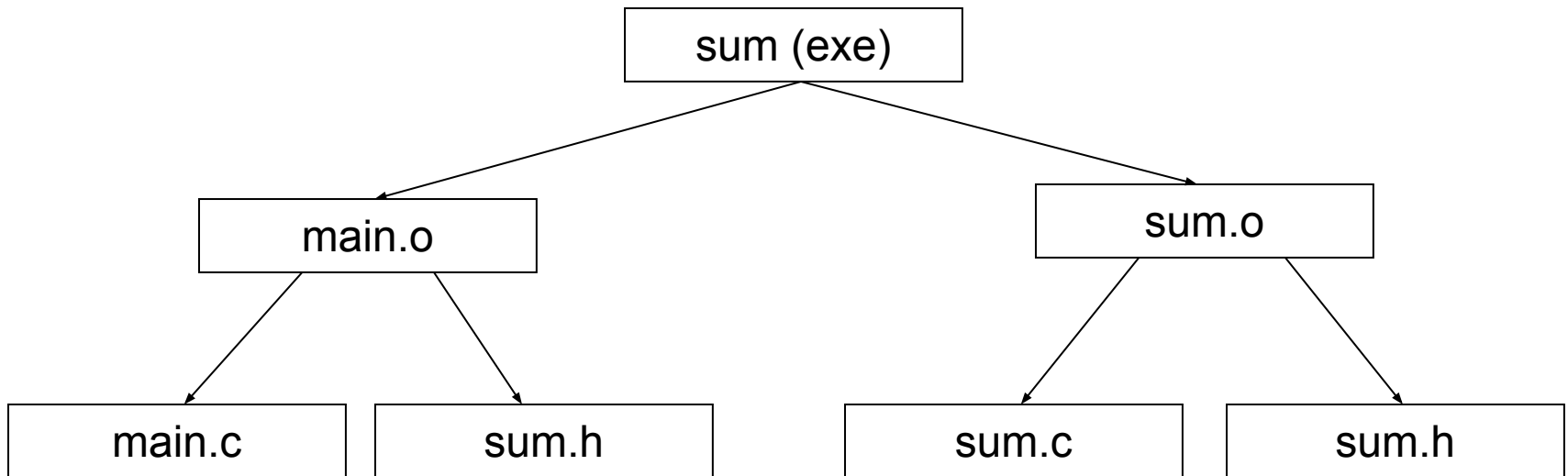
```
sum: main.o sum.o
    gcc -o sum main.o sum.o
```

```
main.o: sum.h
    gcc -c main.c
```

```
sum.o: sum.h
    gcc -c sum.c
```

Project Structure

- Project **structure and dependencies** can be represented as a **DAG** (Directed Acyclic Graph).





Make Operation

- make operation ensures **minimum compilation**, when the project structure is written properly.

- **Do not** write something like:

```
prog: main.c sum1.c sum2.c
```

```
gcc -o prog main.c sum1.c sum2.c
```

which requires **compilation of the full project** when something is changed.



The Last Note

- We process the rules to see if there is a target that should be created or **recreated**.
- Recreation is the case when the **target file is older than one of its dependencies**.

Make Operation - Example

➡ sum: main.o sum.o
➡ gcc -o sum main.o sum.o

➡ main.o: main.c sum.h
➡ gcc -c main.c

➡ sum.o: sum.c sum.h
gcc -c sum.c

<u>FILE</u>	<u>LAST MODIFIED</u>
sum	10:03
main.o	09:56
sum.o	09:35
main.c	10:45
sum.c	09:14
sum.h	08:39

Reference

- Good tutorial for makefiles:

<http://www.gnu.org/software/make/manual/make.html>



Git Tutorial

What is Git?

- Git is a source code version control system. It allows you to:
 - Revert selected files back to a previous state.
 - Revert the entire project back to a previous state.
 - Compare changes over time.
 - See who last modified something that might be causing a problem.
 - And more.

When Git is most useful?

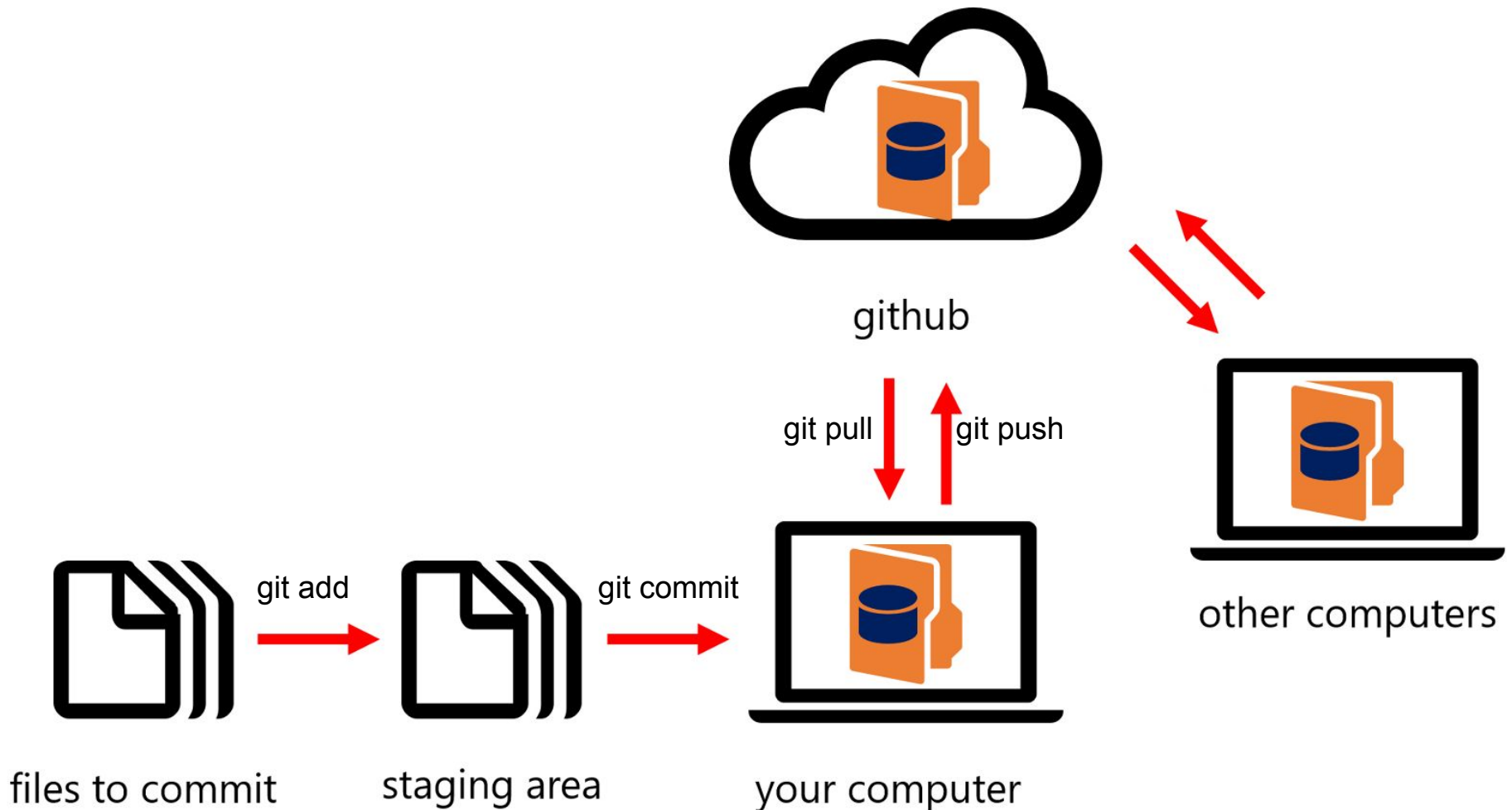
- When you work in a team.
- When you work alone and you want to keep track of the changes you have made.

Git vs Github

- Git is an application installed on your computer.
- Github is hosting git repositories on a server.

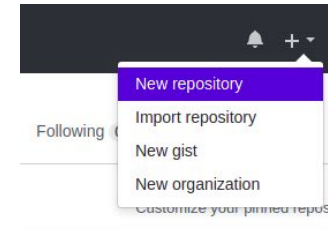


How does git work?



Create a Repository (Github account)

1. In the upper right corner, next to your avatar or identicon, click **+** and then select New repository



Owner **Repository name**


PUBLIC   **hubot** / 

Great repository names are short and memorable. Need inspiration? How about [petulant-shame](#).

Description (optional)

2.

☒  **Public**
Anyone can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**
This will allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.

Add .gitignore: **None** ▾

Add a license: **None** ▾ 

Create repository

Using Git on command line

- Command line is the only place you can run all Git commands.
- Using command line, you can make changes in your codes on your computer and test them, and then add and commit your changes to your local git.

Setup Git

Download Git

- For OSX: <https://git-scm.com/download/mac>
- For Windows: <https://gitforwindows.org/>
- For Ubuntu:
 - \$ sudo apt install git-all

Configure Your Git Environment

- Your Identity
 - The first thing you should do when you install Git is to set your user name and email address.
 - Every Git commit uses this information

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email
```

```
"johndoe@example.com"
```

Creating a project and using Git

- Create a new directory:
 - `mkdir my_project`
- Go to your directory:
 - `cd my_project`
- Put the directory under git revision control:
 - `git init`

Add your codes

- Add files you have made to “staging” area
 - `git add <file1> <file2> ...`
 - For example: `git add my_code.c`
- “git add” needs to be called every time you alter a file
- It may sound redundant, but this workflow makes it much easier to keep a project organized.



Commit

- `git commit -m "A message describing the changes you have made"`
- Think of "git commit" as save

.gitignore

- Specify files and folders we want to ignore
 - .gitignore file
 - Some common examples are:
 - Compiled source
 - *.com, *.class, *.dll, *.exe, *.o, *.so
 - Packages
 - *.gz, *.iso, *.jar, *.rar, *.tar, *.zip
 - Logs and databases
 - *.log, *.sql
 - OS generated files
 - .Trashes, Thumbs.db, ...

Getting a Git Repository

- You typically obtain a Git repository in one of two ways:
 1. You can take a local directory that is currently not under version control, and turn it into a Git repository, or
 2. You can clone an existing Git repository from elsewhere.

Other things you can do in git

- Cloning an existing repository
 - `git clone as`
- Check the staged and untracked files status
 - `git status`
- Create a new branch
 - `git branch <branch>`
- Checkout to another branch
 - `git checkout <branch>`
- Checkout and create a new branch
 - `git checkout -b <branch>`
- Viewing the commit history
 - `git log`

Pull from and push to your remote repository

- git remote add origin
git@github.com:username/your_remote_repo
- Pushing to your remote repository
 - git push origin <branch>
- Pull changes from remote repository
 - git pull origin <branch>

Pull from and push to your remote repository

- Why not just “git pull” or “git push”
 - git does not know which branch on the remote server to download from or upload to
 - `git branch --set-upstream <remote-branch>` sets the default remote branch for the current local branch.

References

- <https://git-scm.com/book/en/v2>
- <https://guides.github.com/activities/hello-world/>
- <https://medium.com/@abhishekj/an-intro-to-git-and-github-1a0e2c7e3a2f>