Open Source SW & Lab - Summer 2023
2. Reasons for OSS & Git-Basics

Walid Abdullah Al
Computer and Electronic Systems Engineering
Hankuk University of Foreign Studies

Based on:
Pro Git (2022) by Scott Chacon, Ben Straub

Computer Vision Lab
Hankuk University of Foreign Studies

Last lecture

- **OSS**
  - Definition
  - Two types of licenses
  - Development map/process
- **VCS/Git**
  - Why?
  - Three types of VCSs
  - Git vs other VCS
  - Three file states in Git
  - Three sections in Git project
  - Git installation and configuration

Today's outline

- **OSS for different stakeholders**
- **Git basics**

Why OSS?

- **Collaborative development**
  - Better software
  - Faster development
  - Less bugs (more eyeballs)
  - More secure
  - No *just trust me*

OSS advantages for various stakeholders

INDIVIDUAL USERS

Flexibility

- Can mix and match software from different sources
- Can save money on buying or leasing software
- Can avoid vendor lock-in, maintaining choice
- Can look under the hood (*"trust, but verify"*)
- More fun!

OSS advantages for various stakeholders

BUSINESSES

Collaborative Development

- Lowers total cost of development
- Speeds up time to market
- Work is submitted to wider community for criticism, suggestions and contributions
- Upstreaming reduces future costs for new products that reuse code
- Uses well-delineated application programming interfaces (APIs)

Marketing

- Customers know what they are getting - they have confidence in quality, there are no secrets
- Product is seen as part of a large ecosystem of related products
- More flexible, possibly modular construction
- Adoption by larger community can help build customers' confidence about the product's durability and stability

OSS advantages for various stakeholders

EDUCATION

Elementary - High School, Public Systems

- Very large amount of available teaching resources at little or no cost
- Very wide range of areas available for using, operating and system administration, and programming
- Students do not become locked into vendor products
- School systems do not have to pay for expensive software, even at a discount
- Generally lower hardware costs, and easier to use old hardware
- Students are learning the skills they will need in the workforce
- Unleashes student creativity: more fun!

University (all advantages of Elementary - High School, and more)

- Students can study and work on the internals of operating systems, applications and libraries, system administration utilities
- Students are ready to enter the workforce where they are most needed
- Good habits are developed, including how to work with the open source community
- Student work is easy for prospective employers to evaluate, since it's publicly accessible

OSS advantages for various stakeholders

DEVELOPERS

- No need to reinvent everything
- Helps to make good, early decisions on product design
- More eyeballs on code can fix bugs faster
- Suggestions and contributions are provided by a large group of developers
- Great for finding the next job
  - Code is readily available for evaluation
  - Can demonstrate how well you work and play with others
  - Can show how good you are at mentoring and maintaining projects and subprojects
- Know you are not alone!

Quiz

From a business perspective, use of OSS (Select all answers that apply):

A. Lengthens time to market, but is worth it because of improved product

B. Enables use of ingredients from other sources and speeds development

C. Makes marketing more difficult because it is hard to differentiate products

D. Makes marketing easier, as some ingredients are already well-known and trusted

Quiz

For school systems at any level, use of OSS (Select all answers that apply):

A. Should be avoided, as students become confused by choices

B. Can lower costs by letting low-cost or free software be used

C. Can lower costs by letting older hardware be used

D. May cause problems with vendor lock-in

Quiz

OSS is (Select all answers that apply):

A. Insecure, as bad actors can easily see the code and hack it

B. Secure, because many developers can easily see the code, look for problems, and mitigate problems when they are discovered

C. Causing security problems, because developers come from different organizations and companies and do not understand each other's code

D. Insecure, because no one is in charge of security

E. Secure or insecure, depending on the quality and priorities of the project maintainers, but at least users can judge this by open discussion and code inspection

Git: basics

Getting a Git repository

- **Two ways:**
  - From a local directory
  - From an existing Git repository
- **From a local directory**
  - Take a local directory that is not under version control
  - And turn it into a Git-directory
- **From an existing Git repository**
  - You may *clone* an existing Git project to your machine
  - That you want to contribute to
  - (We will learn this in a later lesson)

From a local directory

- **Go to that directory**

for Linux:

```
$ cd /home/user/my_project
```

for macOS:

```
$ cd /Users/user/my_project
```

for Windows:

```
$ cd C:/Users/user/my_project
```

- **Intialize a Git repository**

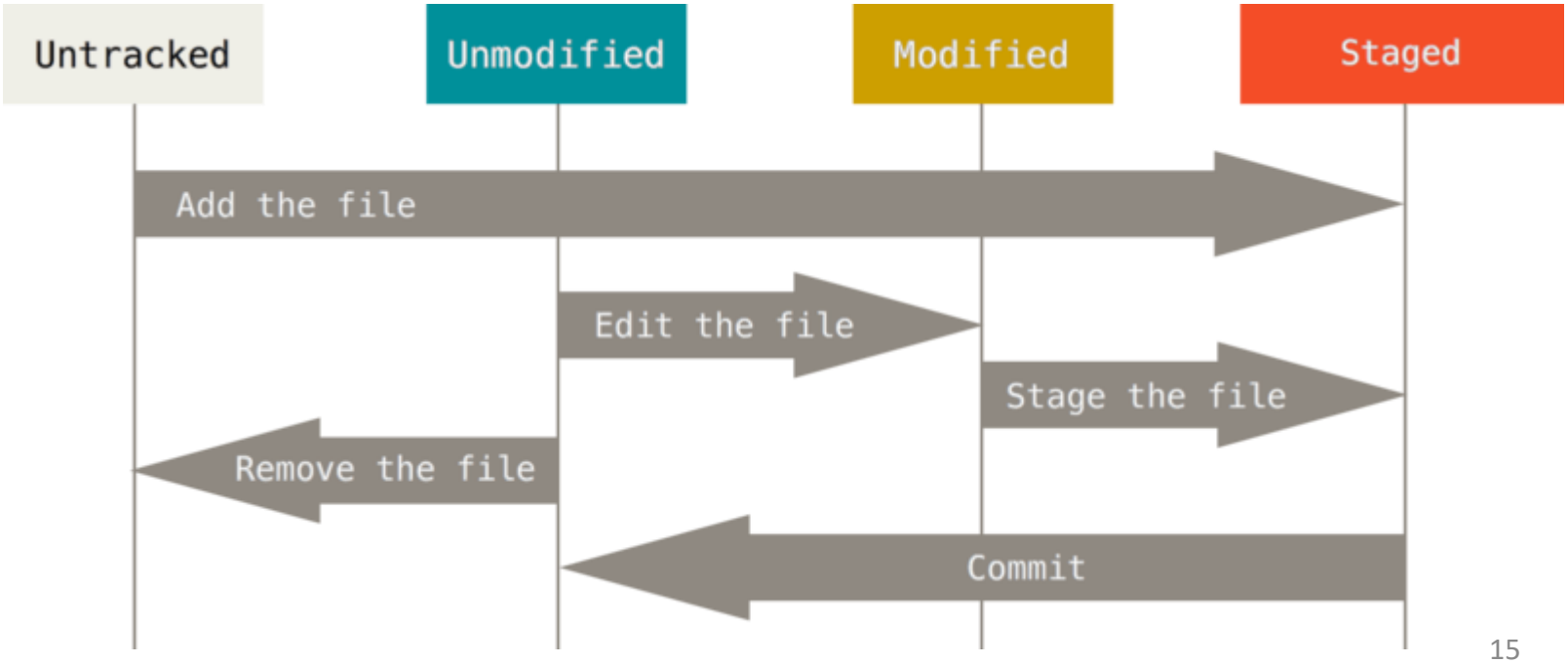
```
$ git init
```

If there are existing files in the directory,
You may start version-controlling existing files
And do an initial commit

```
$ git add *.c
$ git add LICENSE
$ git commit -m 'Initial
project version'
```

File states in a repository

- **Each file can be in one of two states:**
- **Tracked**
  - Unmodified, modified, staged
- **Untracked**



Checking file status

```
$ git status
```

- **Use git status**

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
```

- **a new file created (untracked)**

```
$ echo 'My Project' > README
$ 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)

    README

nothing added to commit but untracked files present (use "git add" to track)
```

Tracking new files

- **Use** `git add`

```
$ git add README
```

- **New file is tracked and staged**

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)

    new file:   README
```

- **Note:** if the added name is a directory, all the files inside are recursively added

Staging modified files

- Assume we have modified a tracked file called `CONTRIBUTING.md`

```
$ 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)

    new file:   README

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:   CONTRIBUTING.md
```

Staging modified files (2)

- **Use `git add` after modification**

```
$ git add CONTRIBUTING.md
$ git status
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
    modified:   CONTRIBUTING.md
```

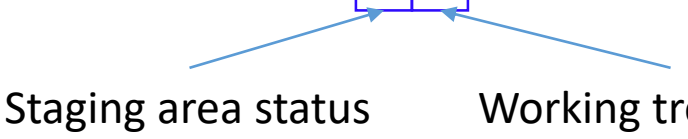
- **You made further changes**
  - CONTRIBUTING.md as both staged and unstaged
  - Should `git add` again

```
$ git status
On branch master
Your branch is up-to-date with 'o
Changes to be committed:
  (use "git reset HEAD <file>..."

    new file:   README
    modified:   CONTRIBUTING.md

Changes not staged for commit:
  (use "git add <file>..." to upd
  (use "git checkout -- <file>...

    modified:   CONTRIBUTING.md
```

Short status

- **Run `git status -s` or `git status –short`**
- **M: modified, A: added, ??: untracked**

```
$ git status -s
 M README
MM Rakefile
A  lib/git.rb
M  lib/simplegit.rb
?? LICENSE.txt
```



Committing your changes

- **`git commit`**
  - Launches editor asking for commit message
- **`git commit –m “my message”`**

```
$ git commit -m "Story 182: fix benchmarks for speed"
[master 463dc4f] Story 182: fix benchmarks for speed
 2 files changed, 2 insertions(+)
 create mode 100644 README
```

- **Note:**
  - Only staged modifications are committed
  - If a file is modified but not staged, this will not be committed or recorded

Auto-staging

- **Skipping the staging step (`git add`)**
- **Use `git commit -a`**

```
$ 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:   CONTRIBUTING.md

no changes added to commit (use "git add" and/or "git commit -a")
$ git commit -a -m 'Add new benchmarks'
[master 83e38c7] Add new benchmarks
 1 file changed, 5 insertions(+), 0 deletions(-)
```

Viewing commit history (`git log`)

- **Example source**

```
$ git clone https://github.com/schacon/simplegit-progit
```

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test
```

```
$ git log --stat
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

 Rakefile | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

```
$ git log --pretty=format:"%h - %an, %ar : %s"
ca82a6d - Scott Chacon, 6 years ago : Change version number
085bb3b - Scott Chacon, 6 years ago : Remove unnecessary test
a11bef0 - Scott Chacon, 6 years ago : Initial commit
```

```
$ git log -p -2
commit ca82a6dff817ec66f44342007202690a9376394
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

diff --git a/Rakefile b/Rakefile
index a874b73..8f94139 100644
--- a/Rakefile
+++ b/Rakefile
@@ -5,7 +5,7 @@ require 'rake/gempackagetask'
 spec = Gem::Specification.new do |s|
     s.platform  =   Gem::Platform::RUBY
     s.name      =   "simplegit"
-    s.version   =   "0.1.0"
+    s.version   =   "0.1.1"
```

Undoing Things

- **Git commit –amend**
  - Overwrites the last commit
- **Git reset HEAD <file>**
  - Unstaging a staged <file>
- **Git checkout -- <file>**
  - Discard modifications in <file>
- **Git restore is also a good option**

Let's do some commits

- **Create an empty directory named "Complex"**
  - In this directory, we want to create a starter Python project for complex number operations
- **Turn this into a Git repository**
- **Create a README.txt file**
  - Explaining the project
- **Track the README.txt file**
- **Commit the changes as 'Initial commit'**
- *(Continue to the next slide)*

Let's do some commits

- **Now create a Complex.py file with the following code**

```
class Complex:
    def __init__(self, re=0, im=0):
        # For a complex number 1+2i, re=1 and im=2
        self.re = re
        self.im = im

    def __str__(self): # how to print the Complex obj
        return str(self.re)+"+"+str(self.im)+"i"

c = Complex(1,2)
print(c)
```

- **Test it**
- **Add Complex.py to tracking**

Let's do some commits

- **Edit the README.txt file**
  - To explain the class members and methods
- **Stage the modified file**
- **Commit as "Basic class defined"**

- **Finally, check the commit history using `git log --stat`**
- **And submit the screenshot of commit log on Eclass**

Done for today!

- Please remember to submit your final screenshot on Eclass