CS265
Advanced Programming Techniques

Git & Github

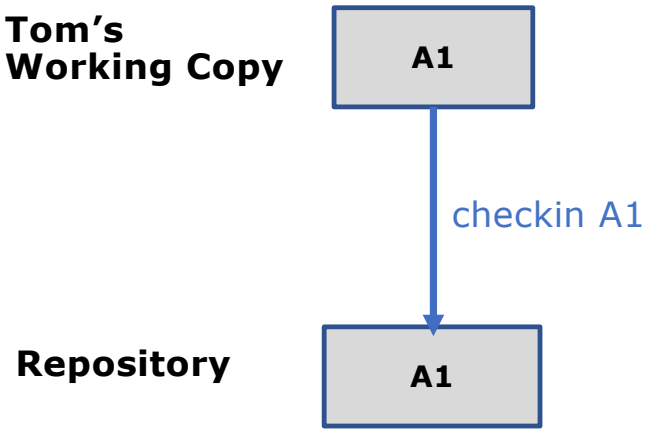
Agenda

- Introduction to version control
- Git
- Github

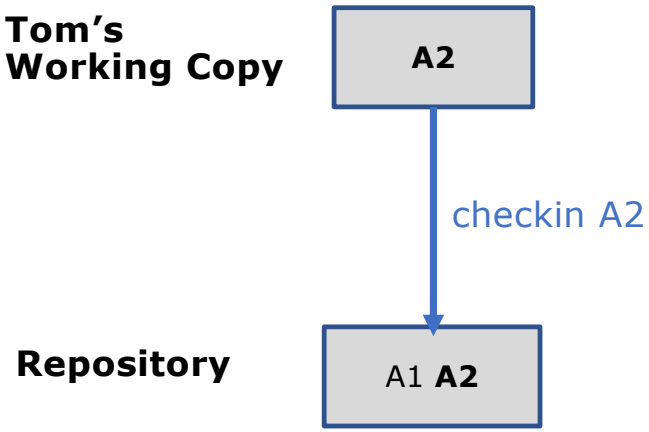
The need for version control

- Version control is a system that
  - records changes to a file or set of files over time
  - allows you to revert selected files to previous versions or entire projects to a previous version
- Why?
- If something goes wrong, you can always recover
  - e.g., a programmers went down the wrong path
  - e.g., a customer likes the previous image/layer of a web design
- Best approach for large teams to collaborate by sharing code in a large projects
  - But we also need a way to manage conflicts

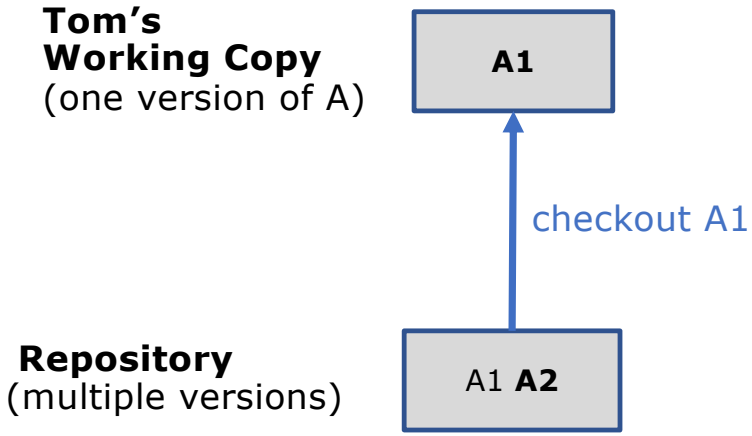
Version control – v1 (Single User)



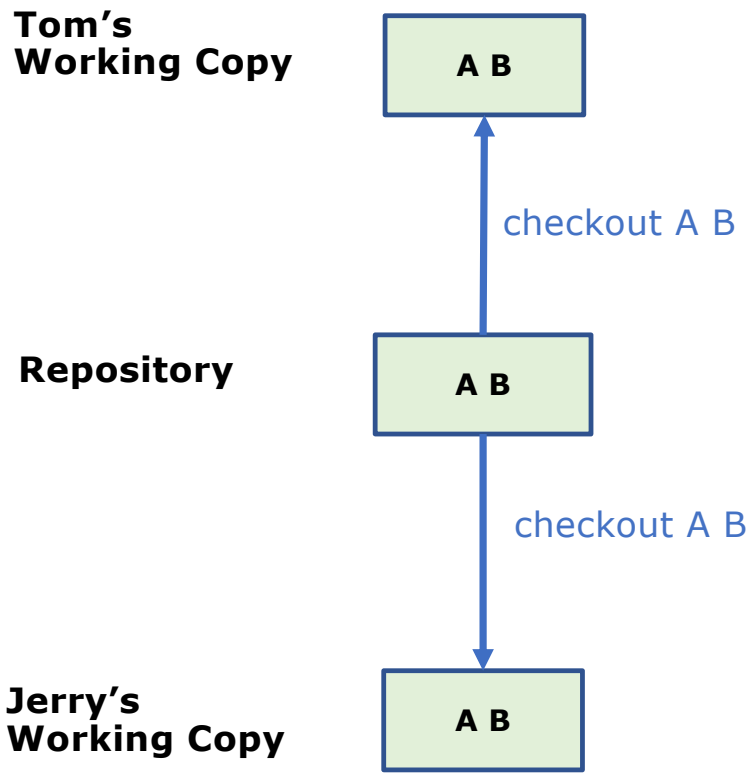
Version control – v1 (Single User)



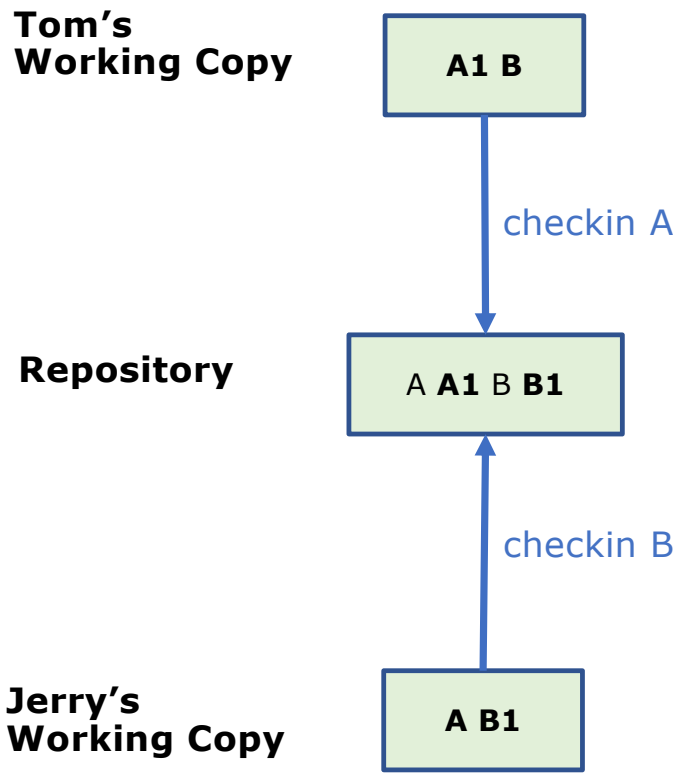
Version control – v1 (Single User)



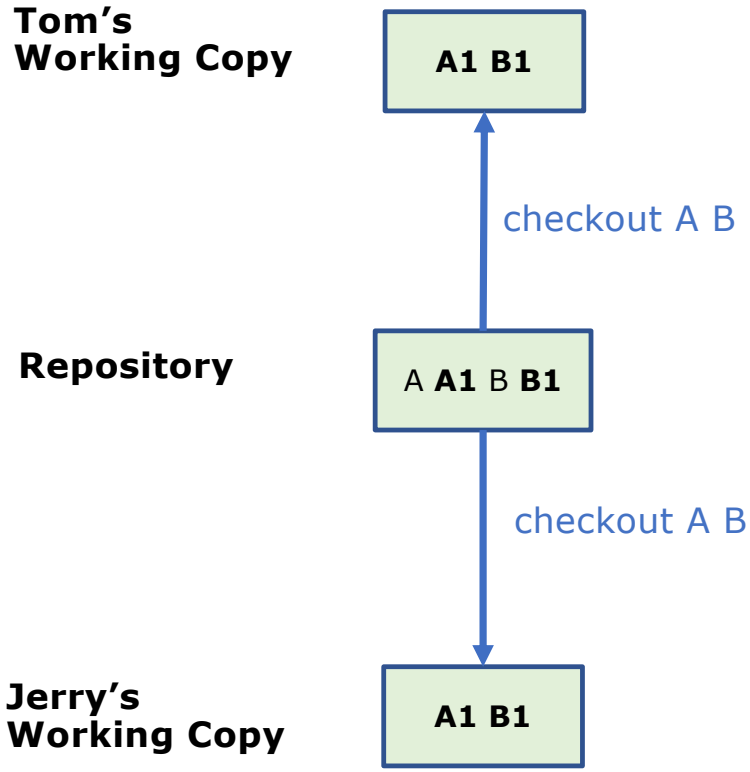
Version control – v2 (Multiple Users)



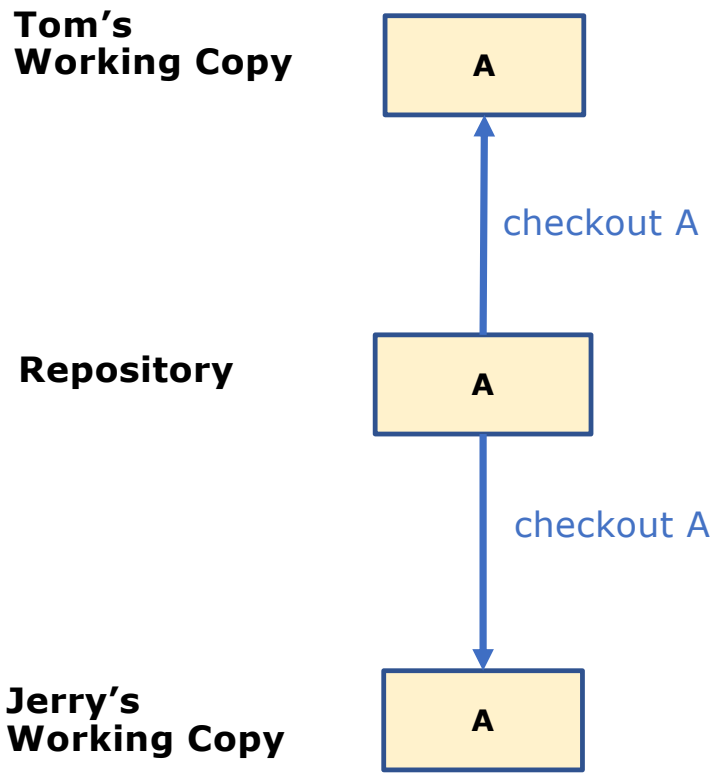
Version control – v2 (Multiple Users)



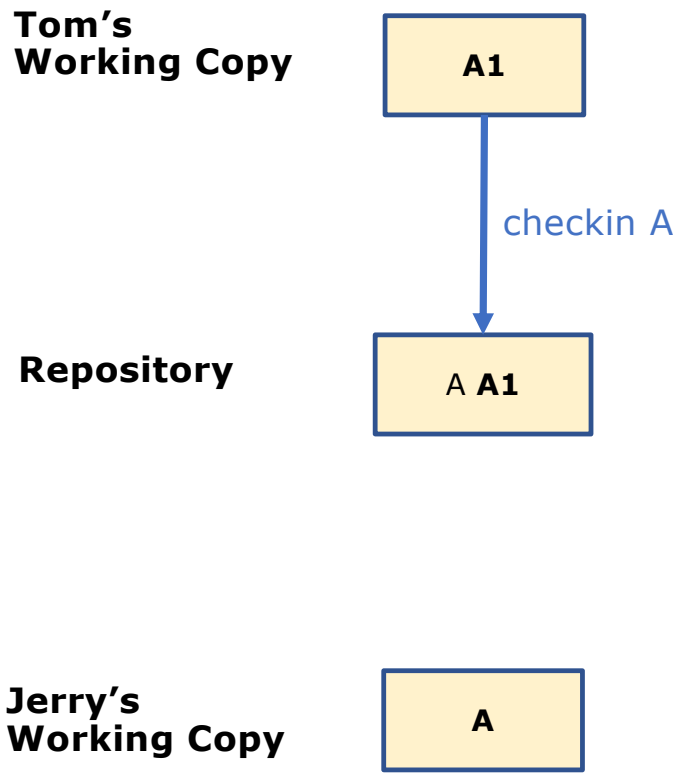
Version control – v2 (Multiple Users)



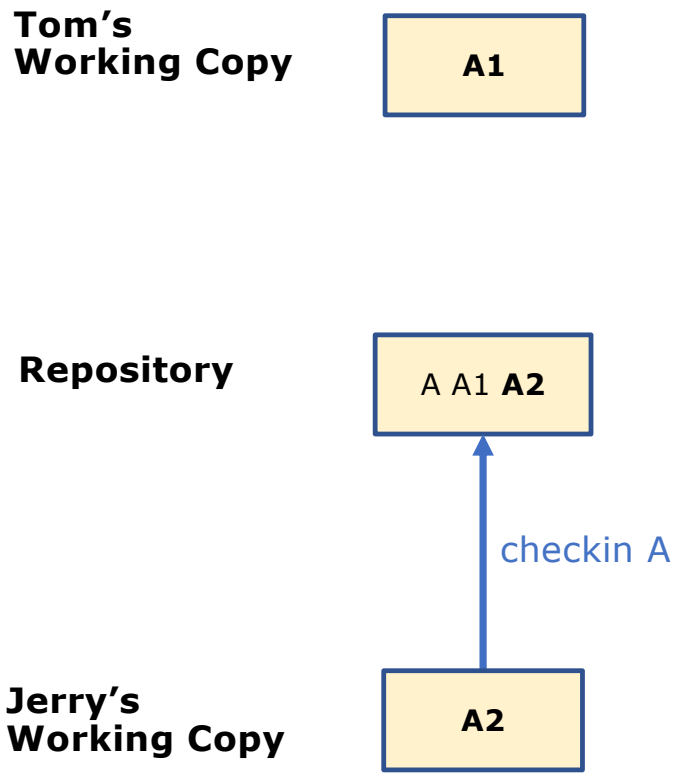
Version control – v3 (Conflicts)



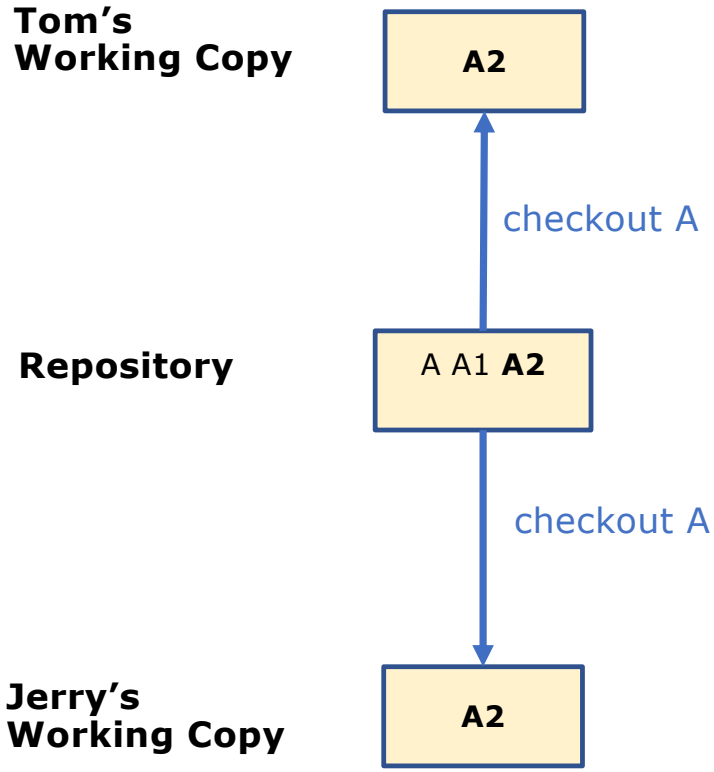
Version control – v3 (Conflicts)



Version control – v3 (Conflicts)



Version control – v3 (Conflicts)



Problems
1) Tom lost his changes
2) Tom's changes are lost in the repository (not the latest)
3) Jerry never show Tom's changes

Git Version Control

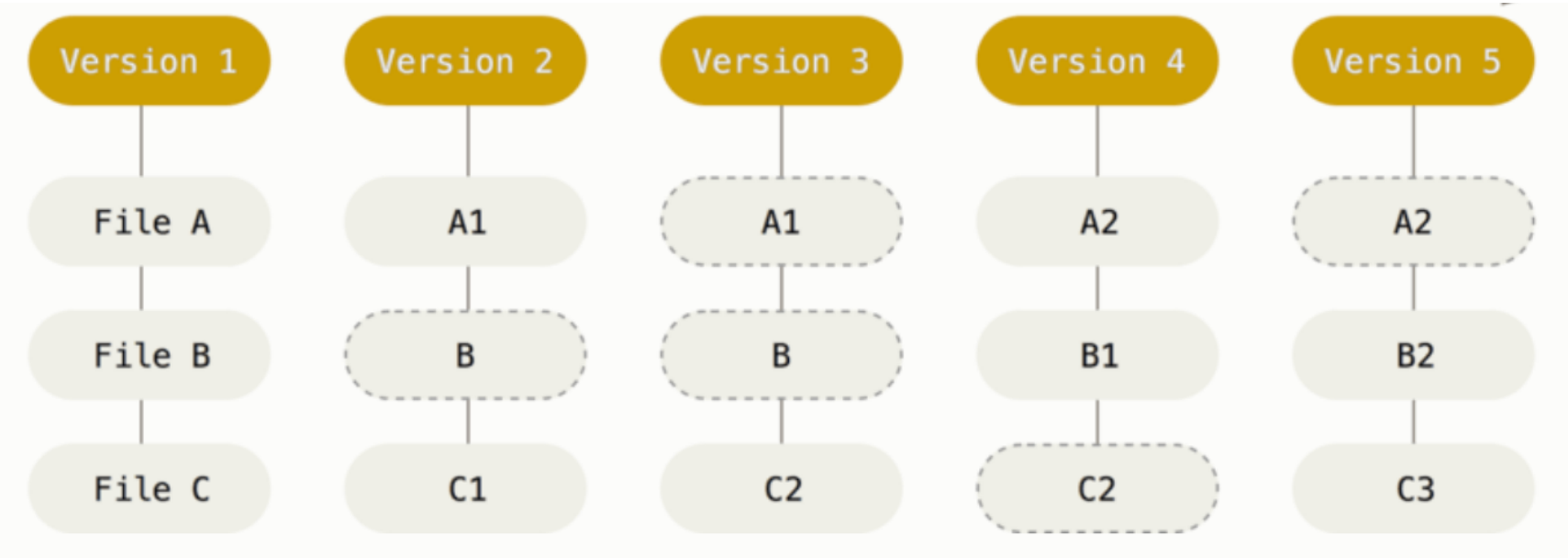
- Git was created by Linus Torvalds in 2005 for development of the Linux kernel
- Tracks the history of a collection of files (source code)
  - Git tracks a repository
- Allows us to:
  - See what files changed, and when
  - Compare (diff) two ore more versions
  - Recover (check out) older versions of files
  - Experiment with new ideas, features, without the risk of losing existing work (branching)
- Greatly facilitates collaboration

Git repository

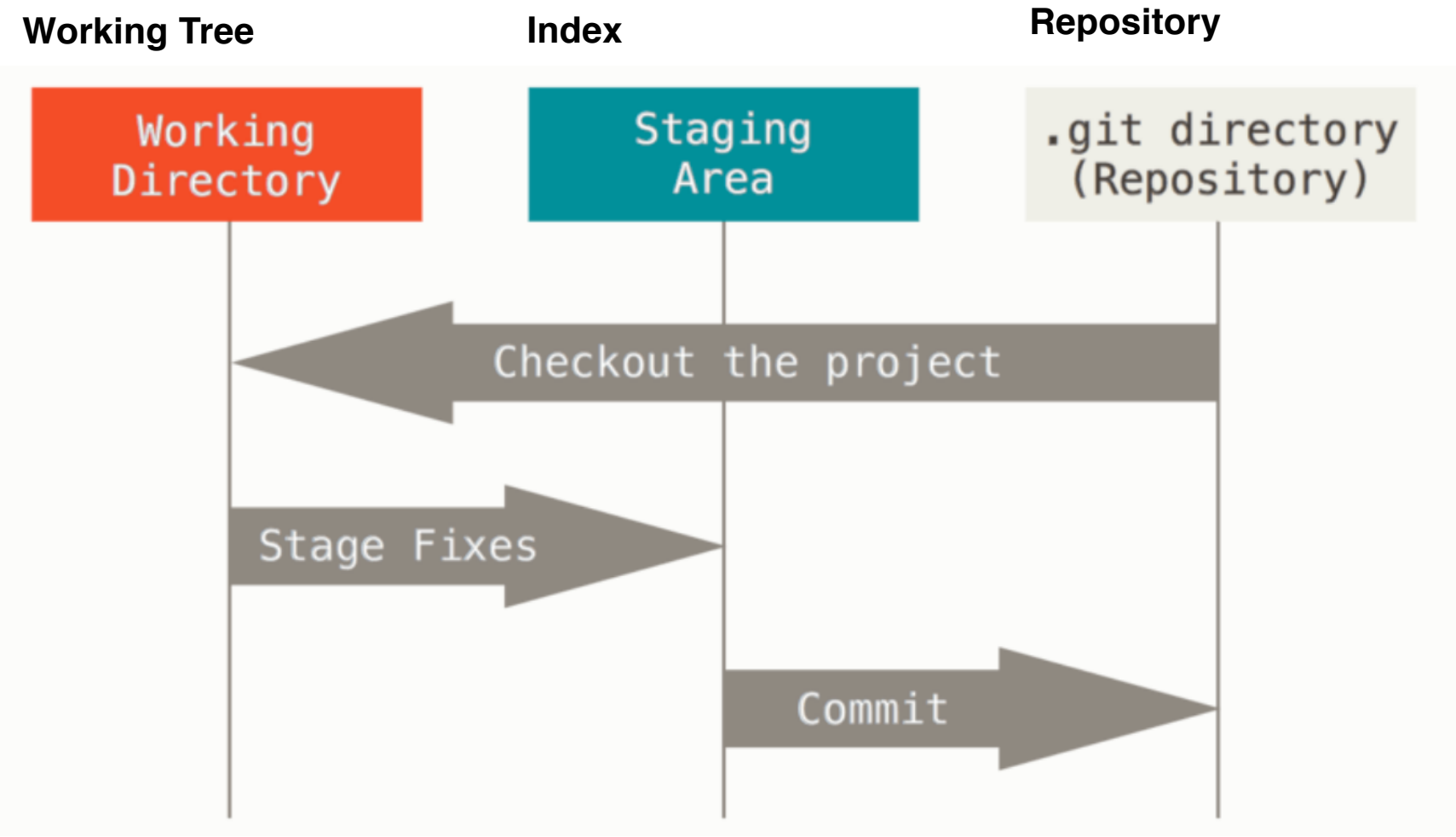
- A container for a project that needs to be tracked
- Can have many different repositories for many different projects

Git stores data as snapshots

- Git does not store changes (deltas)
- It stores complete snapshots of every file with every “save/commit”
- If a file has not changed, to save space, Git uses a link to the previous full version

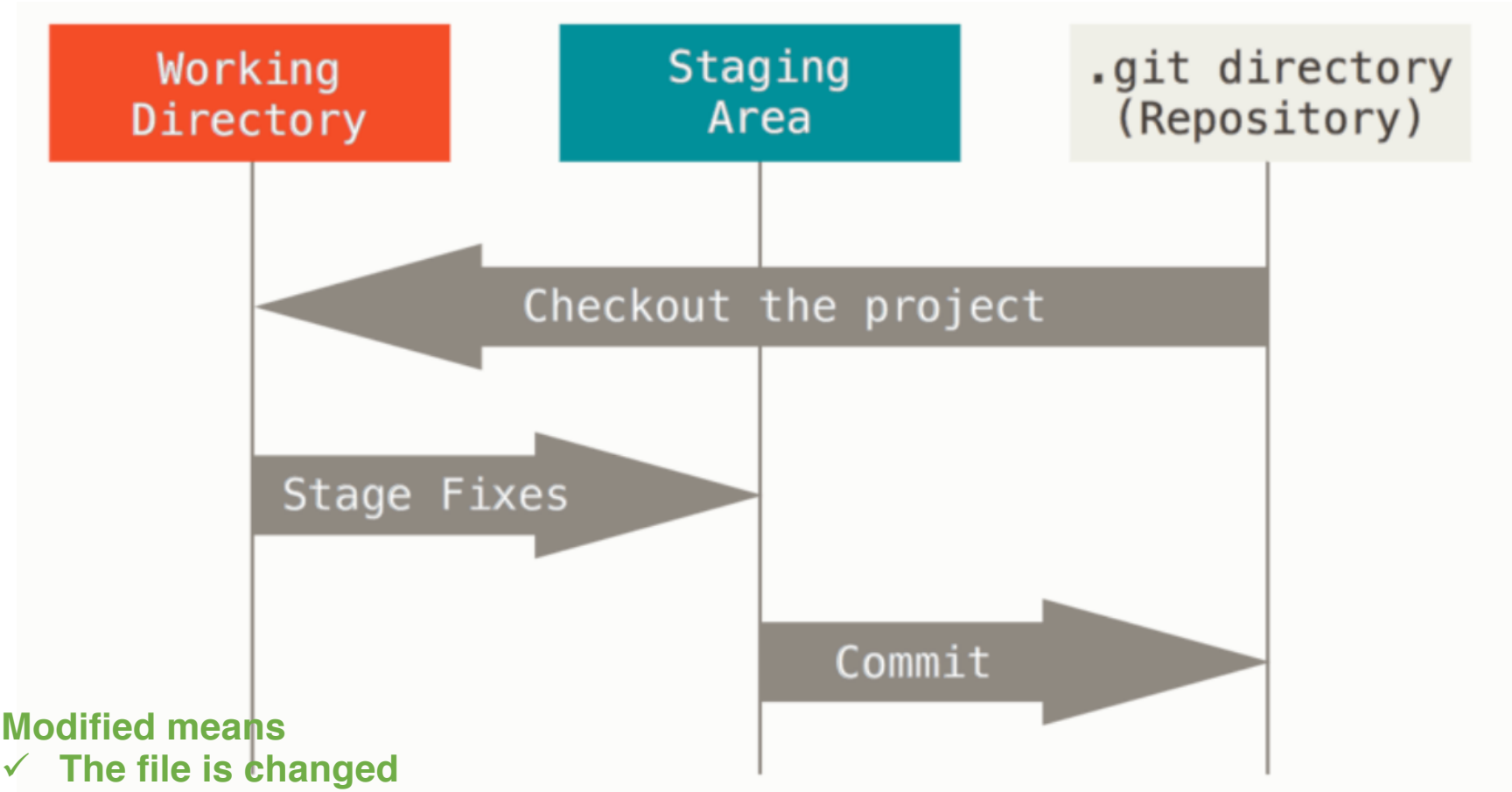


Working Tree vs Index vs Repository



File stages: Modified, Staged and Committed

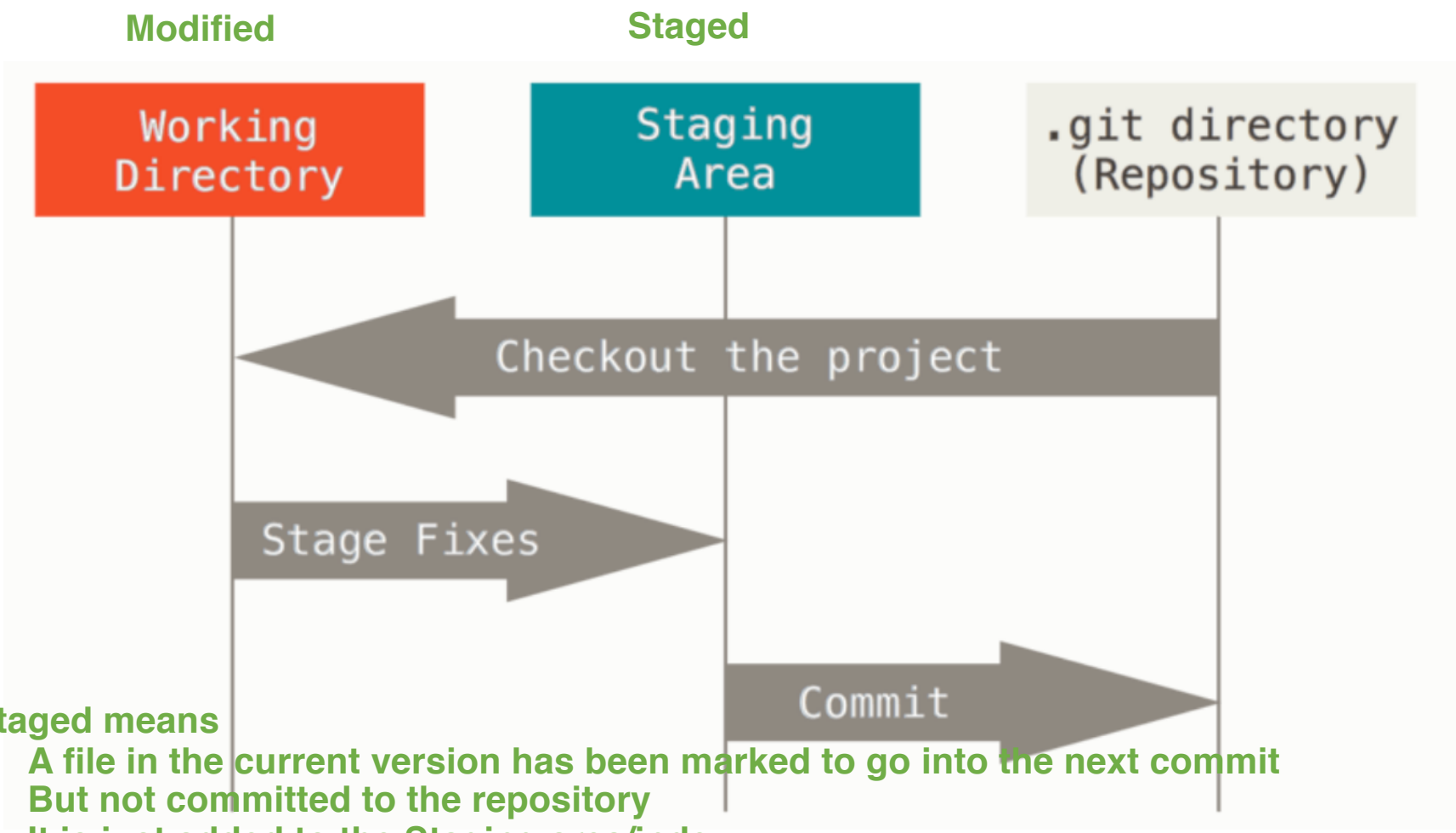
Modified



Modified means

- ✓ **The file is changed**
- ✓ **But not committed to the repository**

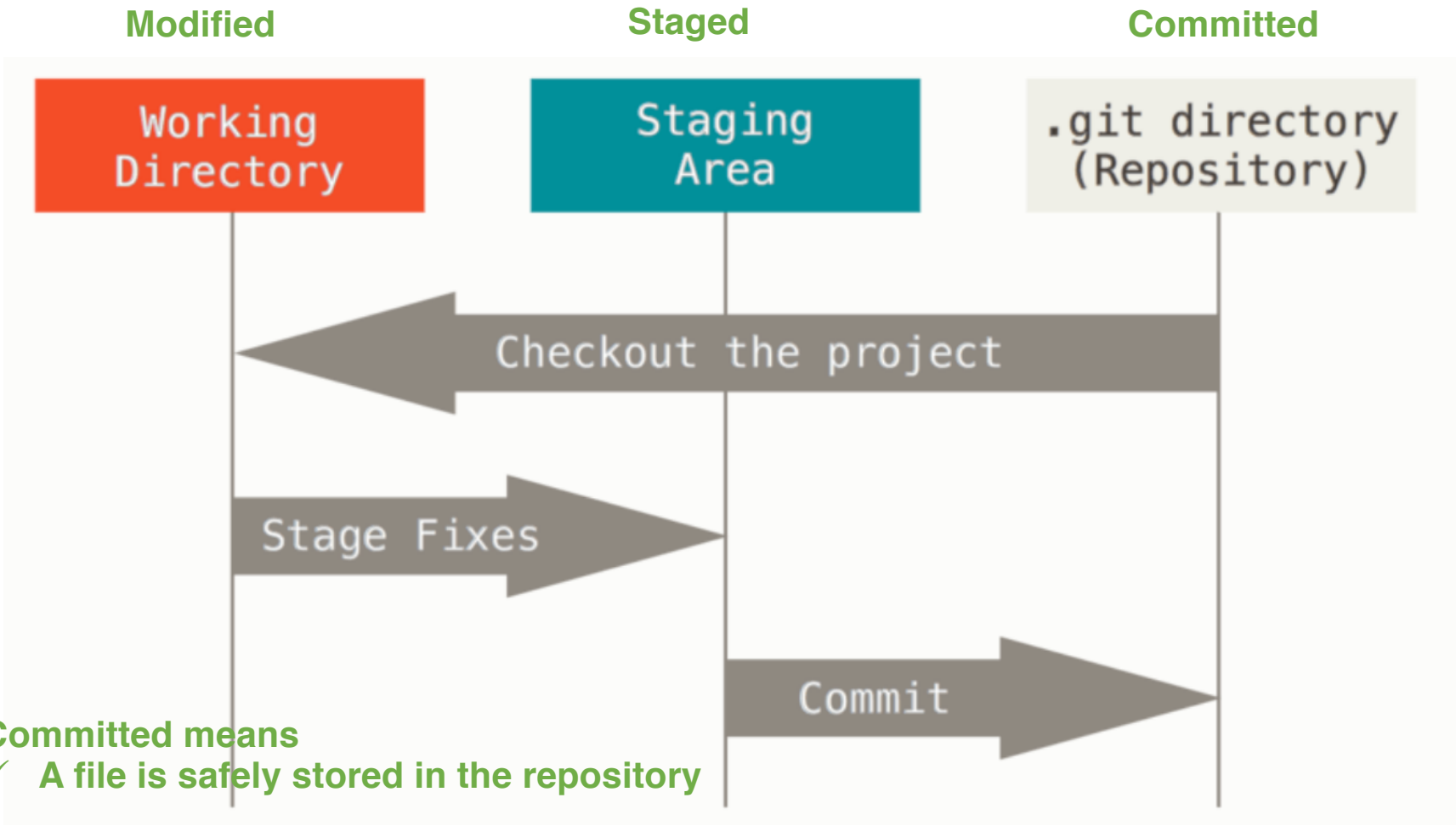
File stages: Modified, Staged and Committed



Staged means

- ✓ A file in the current version has been marked to go into the next commit
- ✓ But not committed to the repository
- ✓ It is just added to the Staging area/index

File stages: Modified, Staged and Committed



Committed means

- ✓ **A file is safely stored in the repository**

Index vs Working Tree

- A working tree is just a copy of the files checked out of the repository. It's where you work on your files
- The index is the set of changes to be committed.
- The index may be different from the working directory
- Changes to the working directory must be added to the index

`git config` – Setting up git

- You can have multiple repositories, in various locations
- The entire subtree is included
- Use `git config` to set some global settings (~/.gitconfig)

```
$ git config --global user.name dv35
$ git config --global user.email dv35@somewhere.com
$ cat ~/.gitconfig
[user]
        name = dv35
        email = dv35@there.com
```

Configuration Files

`/etc/gitconfig` global system
`~/.gitconfig` in the users home directory
`.git/config` in the repository directory

`git init` – Create a repository

Create an empty Git repository or reinitialize an existing one with `git init`

```
$ mkdir lab-git
$ cd lab-git
$ git init
Initialized empty Git repository in /home/…//lab-git/.git/
$cd .git
ls
branches/  config  description HEAD  hooks/  info/  objects/  refs/
```

`git status` - Check the status of a repository

- Use `git status` to show the working tree status

```
$ git status
On branch master


No commits yet


nothing to commit (create/copy files and use "git add" to track)
```

Adding a new file

- Let's create a new file

```
$ echo 'Hello, Wrld!' > hello.txt
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

hello.txt

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

`git add` - Adding a new file to the index

- Use `git add` to add file contents to the repository
- Any changes need to be staged, or added to the index
- commit will add staged changes to the repository, clear the index

```
$ git add hello.txt
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

new file:   hello.txt
```

`git commit` – Record changes to the repository

- We can now commit

```
git commit [-m msg ]
```

- Note, the `-m`. It'll ask you for a message anyway, so might as well do it now
- Just a quick msg, help you distinguish between commits

```
$ git commit -m "Initial commit"
[master (root-commit) 199e44c] Initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt
```

Oops..

- Let's fix that error

```
$ sed -i s/Wrld/World/ hello.txt
$ git status
On branch 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:   hello.txt


no changes added to commit (use "git add" and/or "git commit -a")
```

Commit new changes

- We can add individual files as before

```
git add hello.txt
```

- Or, the -u flag will have git search all the files it's previously seen (been added) for updates. This will not pull in any new files

```
git add –u
```

- Or, to stage everything

```
git add .
```

- Now, we can commit

```
git commit –m"Fixed typo"
```

Examining the repository - log

```
$ git log
commit dfd036c62dc26c39ee2c43943e0d02b31a4e8675 (HEAD -> master)
Author: dv35 <dv35@somewhere>
Date:   Tue Mar 10 13:15:28 2020 -0400

    Fixed typo

commit 199e44c0869bb848d0bbd79012ac1d2c22e9739a
Author: dv35 <dv35@somewhere>
Date:   Tue Mar 10 13:07:40 2020 -0400

    Initial commit
```

Examining the repository - log

- To see a more compact version

```
$ git log --oneline
dfd036c (HEAD -> master) Fixed typo
199e44c Initial commit
```

Examining the repository - diff

- To see the difference between two commits

```
$ git diff 199e44c dfd036c
diff --git a/hello.txt b/hello.txt
index e9c608e..8ab686e 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1 +1 @@
-Hello, Wrld!
+Hello, World!
```

- Can, optionally, indicate an individual file:

```
$ git diff 199e44c dfd036c -- hello.txt
```

Git commands

`Git init`	Initialize a new repo
`Git status`	Check status of working tree
`Git add` *`files`*	Add files (changes) to index
`Git commit -m`*`msg`*	Commit changes in index to repo
`Git log`	See commit history
`Git diff`	Compare versions
`Git help`	Get help

SHA1 – commit identifiers

- Each revision (commit) gets a unique identifier

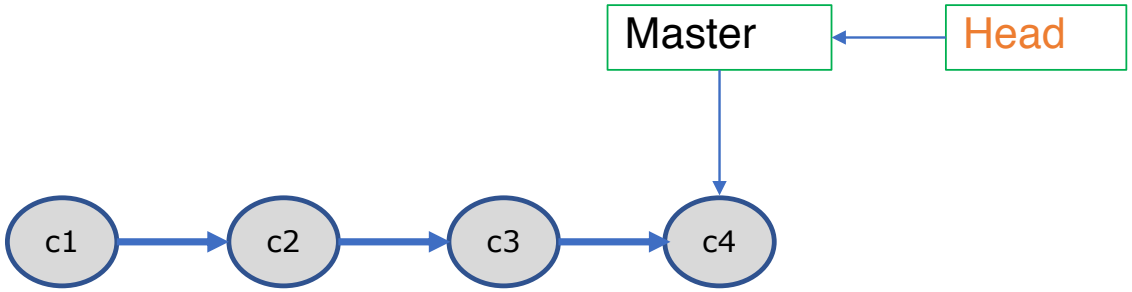
```
$ git log
commit dfd036c62dc26c39ee2c43943e0d02b31a4e8675 (HEAD -> master)
Author: dv35 <dv35@somewhere>
Date:   Tue Mar 10 13:15:28 2020 -0400

    Fixed typo

commit 199e44c0869bb848d0bbd79012ac1d2c22e9739a
Author: dv35 <dv35@somewhere>
Date:   Tue Mar 10 13:07:40 2020 -0400

    Initial commit
```

SHA1 – commit identifiers (cont'd)

- A hash value, created from the committed content, plus a header
  - Note this serves as a padlock. The committed content can't be modified.
- Each serves as a way to identify individual commits
- You can simply use the first 4 characters (providing that they make a prefix unique to the repository)

Git Commit History

- The first commit in a repository creates a default MASTER branch
- The HEAD is a reference, points to the current checked point

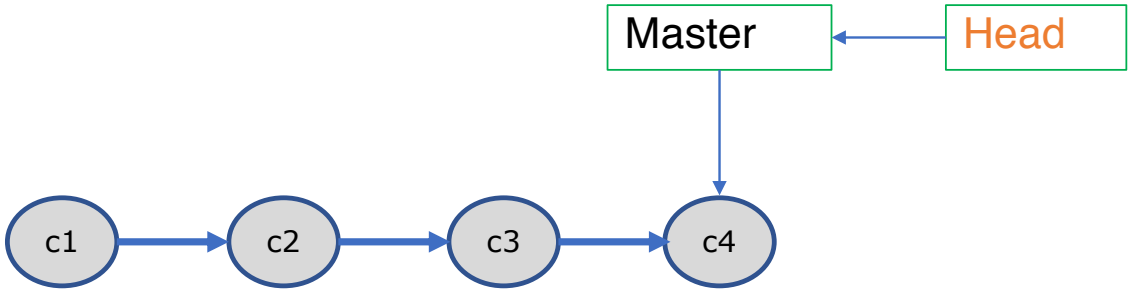


Undoing things

- Checkout Commit (safe)
  - Moves the HEAD pointer to a specific commit point or a specific switch between branches
  - It rollbacks any content changes to those of the specific commit
  - Will not make changes to the commit history
  - Has potential to overwrite files in the working directory
- Revert Commit (still safe)
  - Rollback changes that have been committed
  - Creates a new commit, that undoes all the changes performed
  - It does not delete the original commit from the commit history
- Reset Commit (potentially unsafe)
  - Reset the entire working tree to the last committed state
  - Remove uncommitted changes (ok to do)
  - Hard reset to undo the commit, as if it never happened (potentially not ok to do)

Git Branches

- A branch is an independent line of development
  - Each developer in a collaborative environment has her own branch
  - New branches can be created for bug fixes or for new features
- The first commit in a repository creates a default MASTER branch
- The HEAD is a reference, points to the current (checked out) branch

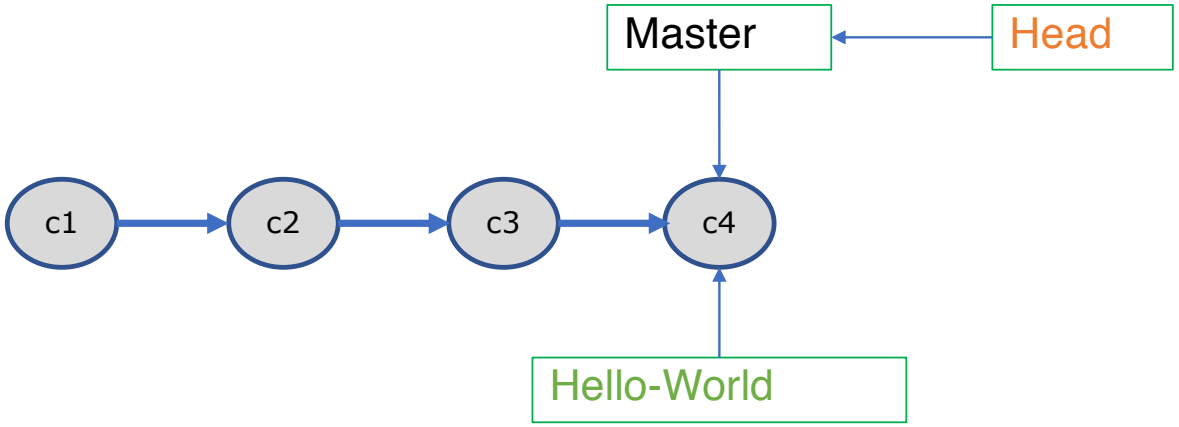


Git Branches – `git branch`

- You can create a new branch named "Hello-World"

```
git branch Hello-World
```

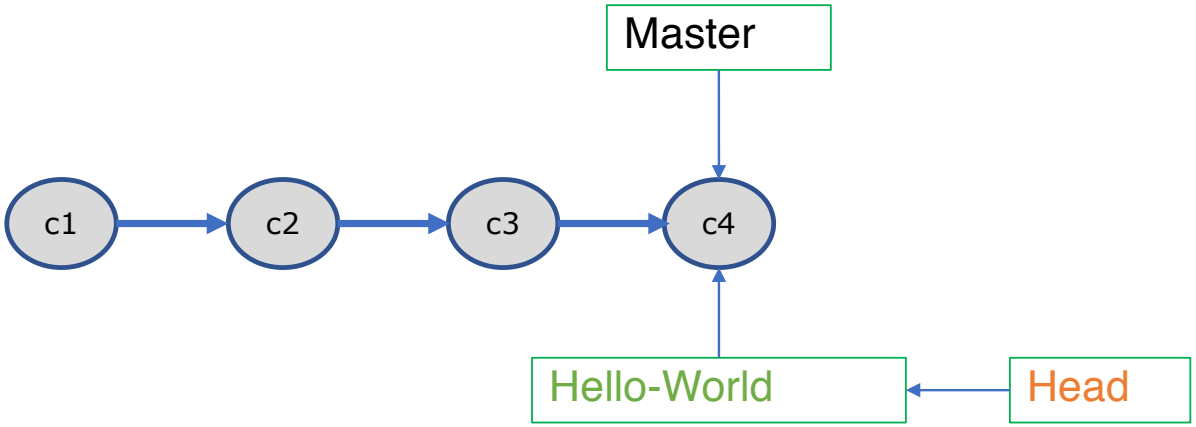
- This only creates the new branch



Git Branches – `git checkout`

- Use git checkout to switch to the new branch

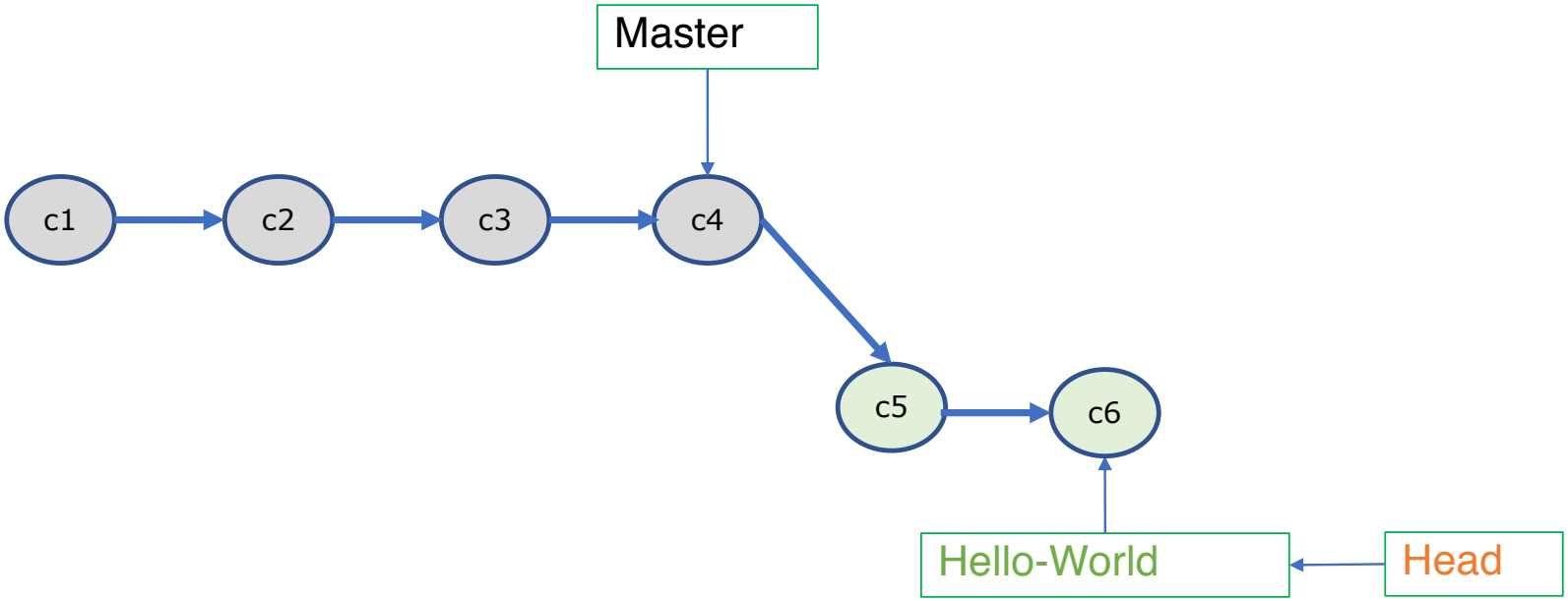
```
git checkout Hello-World
```



Git Branches – `git commit`

- Use git add and git commit to makes changes to the new branch

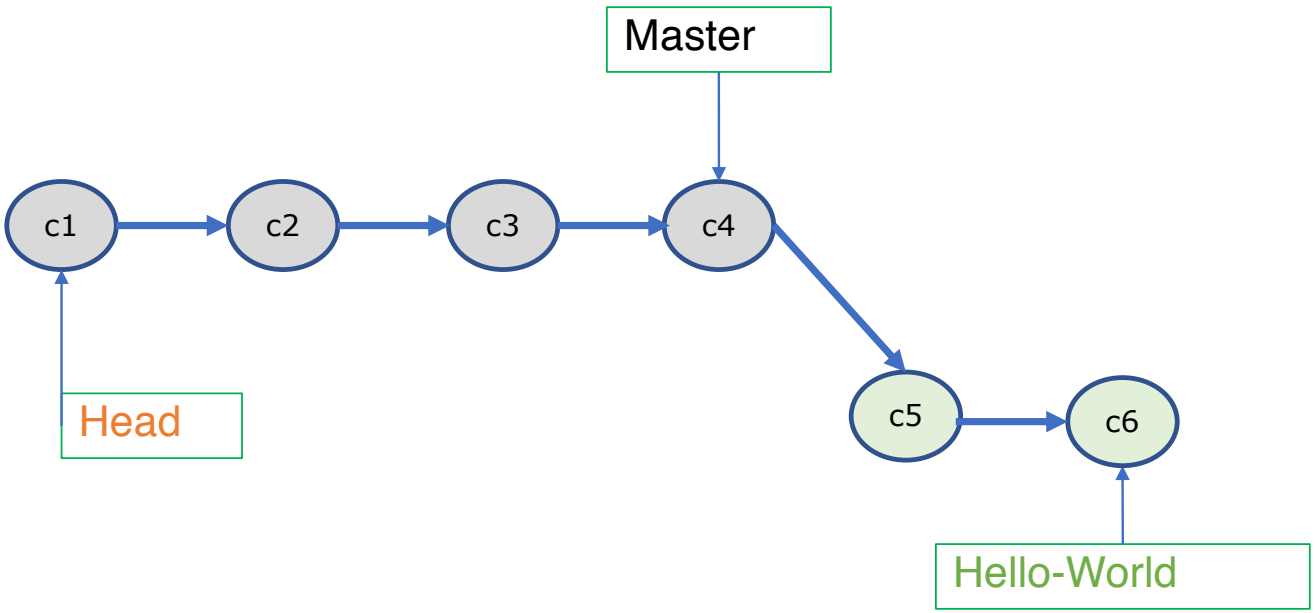
```
git commit -m "commit c5"

git commit -m "commit c6"
```



Git Branches – detach head

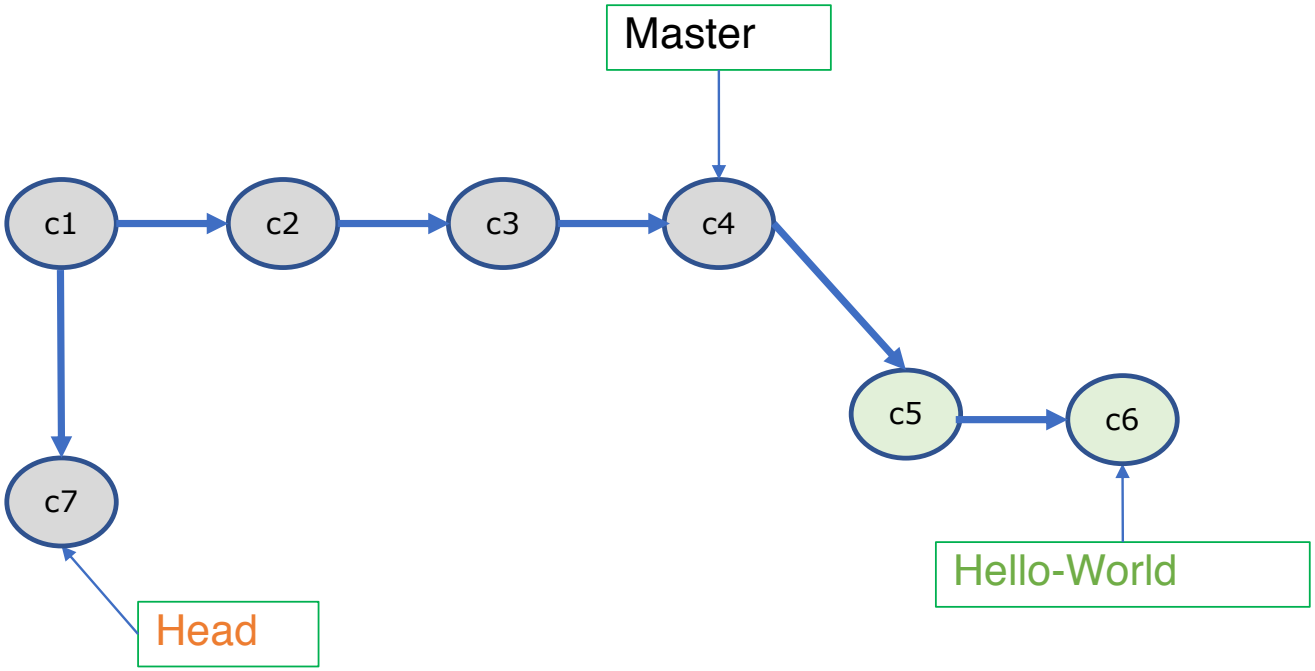
- Use git checkout to switch to a different commit point

```
git checkout c1
```



Git Branches – detach head

- Any new commits will start from here
- Be careful – moving the head does not create a new branch
- Always use branches when you are solving new problems

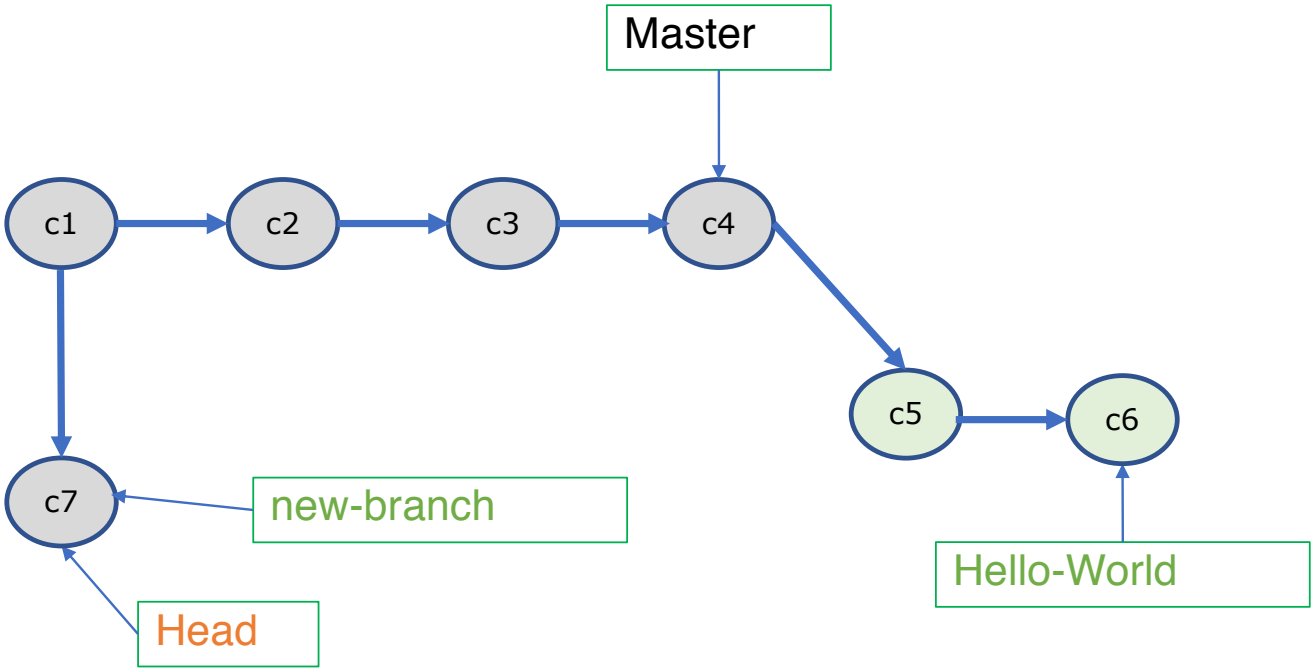


Git Branches – detach head

- Always use branches when you are solving new problems

```
git checkout -b "new-branch"
```

- Can always go back **with** `git checkout master`



Git Branches support the full development lifecycle

Git Brach commands

`git branch -a` lists all branches (* next to the current one)

`git checkout branch <name>` switch to a another branch

`git branch -D <name>` delete a branch

`git checkout -b <name>` checkout and switch to another branch in one step

Branches - example

```
$ git checkout master
Switched to branch 'master'
$ git checkout -b “feature-1”
Switched to a new branch 'feature-1'
$ echo ”f1” > f1.txt
$ git add .
$ git commit -m “added f1”
[feature-1 e7da0a5] added f1
 1 file changed, 1 insertion(+)
 create mode 100644 f1.txt

$ git checkout master
Switched to branch 'master'
$ git checkout -b “feature-2”
Switched to a new branch 'feature-2'
$ echo ”f2” > f2.txt
$ git add .
$ git commit -m “added f2”
[feature-2 5b0325f] added f2
 1 file changed, 1 insertion(+)
 create mode 100644 f2.txt
$ git branch –a
  feature-1
* feature-2
  master
```

Merge

- Use `git merge` to join two branches together

- You must be on the branch you need to merge into

```
git checkout master
```

- You can use a merge command

```
git merge <branch-name>
```

Merging branches

```
$ git checkout master
Switched to branch 'master'
$ git merge feature-1
Updating dfd036c..e7da0a5
Fast-forward
 f1.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 f1.txt
$ git merge feature-2
Merge made by the 'recursive' strategy.
 f2.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 f2.txt
```

Note that the algorithm used to merge the second branch is by "recursive strategy". This is because the master changed since the branch "feature-2" was created

Github

- Created in 2008 by Chris Wanstrath, P.J Hyett, Tom Preston-Werner, and Scott Chacon
- Owned by Microsoft
- It is a service that hosts remote repositories
- People collaborating on the same remote repository can
  - Clone the remote repository in their own computers
  - Develop on their own computers
  - Can push/pull code

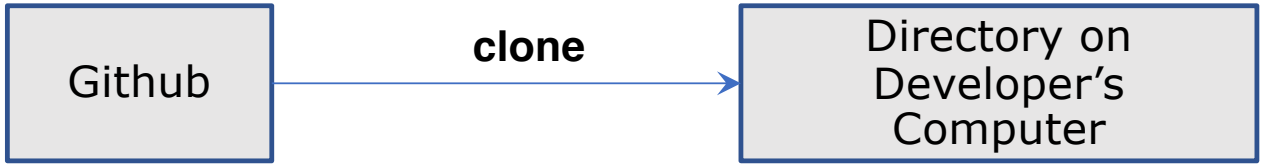
GitHub.com

- Sign in or sign up
- Two possible initial steps to start working with GitHub

1. Clone

2. Push

Git/GitHub - `Clone`



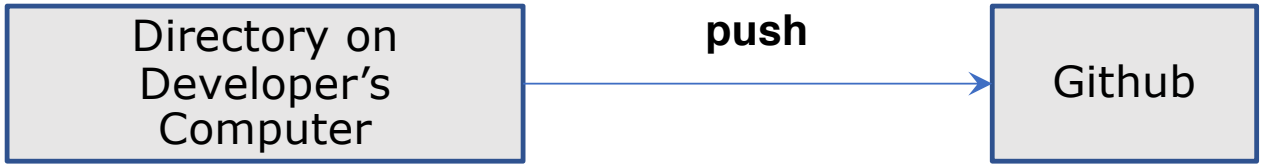
- You can create a new repository on GitHub and clone it into your local repository

```
git clone <remote-repository>

git remote -v
```

e.g.

```
git clone https://github.com/dxvist/CS571-Demo.git
```

Git/GitHub - Push



- Or you can create a remote repository in Github say

  `https://github.com/dxvist/cs265-demo`

- And push a local repository into the remote Github repository using

  `git push <remote-repository> <branch>`

  e.g.

  `git push https://github.com/dxvist/cs265-demo master`

GitHub.com - alias

- You can create an alias for the remote repository

```
git remote add origin  https://github.com/dxvist/cs265.git
git push origin master
```

GitHub.com capabilities

- Getting & Creating Projects
  - Initialize a local git repository with `git init`
  - Clone a remote repository locally
- Commit history & repository status
- Branching & Merging
  - List branches, create branches, delete branches, checkout to branches
  - Merge a branch into an active branch
- Sharing and Updating Projects
  - Pushing data from a local repository to the remote repository using

    ```
    git push origin master
    ```

  - Pulling data from the remote repository locally using

    ```
    git pull origin master
    ```

- Inspection and Comparison with `git log` and `git diff`

Lessons

- Lesson 1: Software development requires change management
- Lesson 2: Git/Gihub are the world's leading version control mechanism

Resources

- These notes
- The entire Git Pro book https://git-scm.com/book/en/v2

Acknowledgements

These slides are copied or inspired by the work of the following courses and people:

- CS265 & CS571, Drexel University, Kurt Schmidt, Jeremy Johnson, Geoffrey Mainland, Spiros Mancoridis, Vera available at https://www.cs.drexel.edu/~kschmidt/CS265/