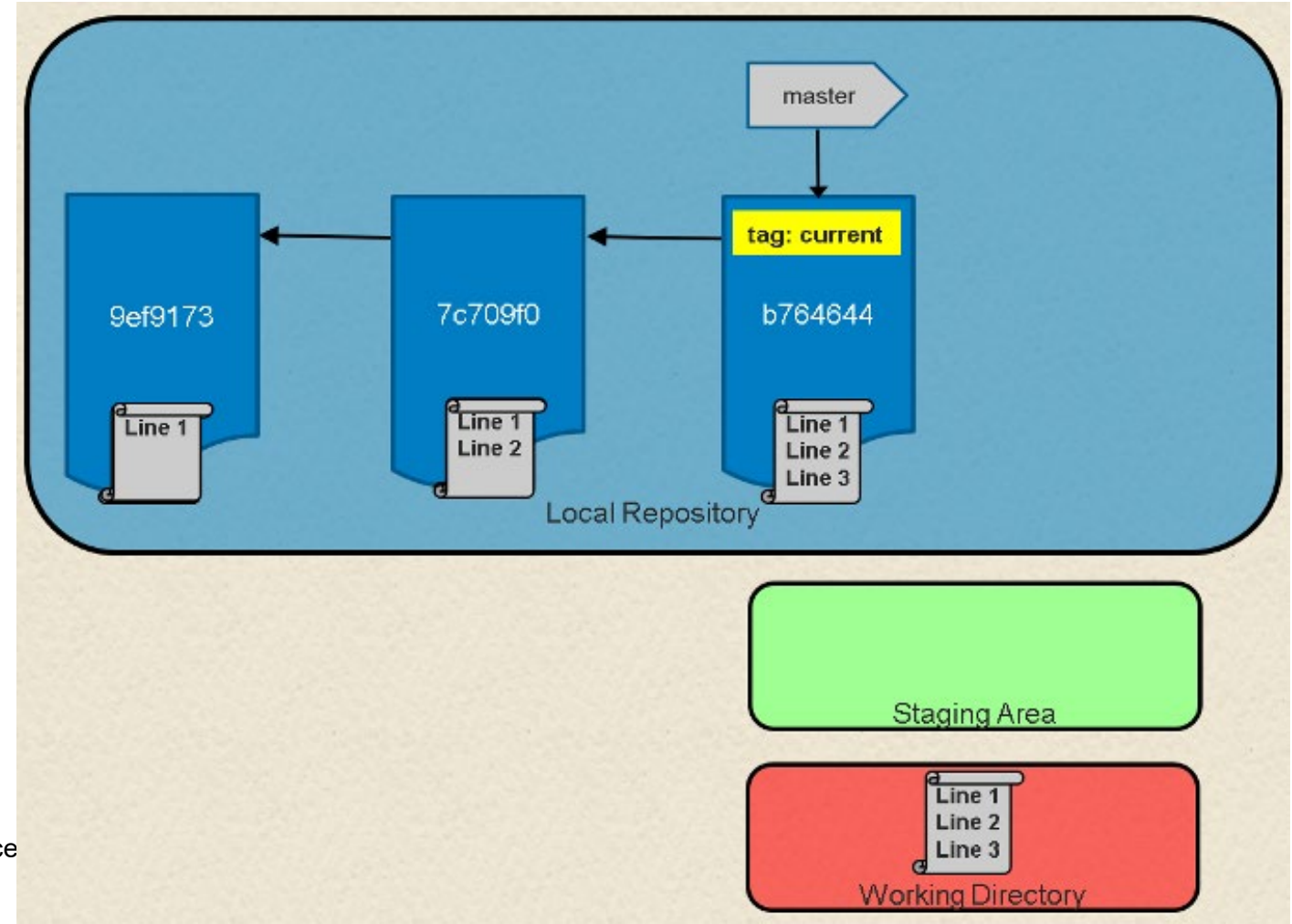
Version Control Revisited

- Basic review after initial use
- Status, diffs and logs
- Some more details about git workflow

Example of current state

- $ git log --oneline
- b764644 File with three lines
- 7c709f0 File with two lines
- 9ef9173 File with one line



Adding & Committing files

1. The first time we ask a file to be tracked, *and* **every time before we commit a file** we must add it to the staging area:

```
$ git add README.txt hello.java
```

This takes a snapshot of these files at this point in time and adds it to the staging area.

Note: To unstage a change on a file before you have committed it:

```
$ git reset HEAD filename
```

2. To move staged changes into the local repo we commit:

```
$ git commit –m “Fixing bug #22”
```

Note: You can edit your most recent commit message (if you have not pushed your commit yet) using: `git commit --amend`

Note: These commands are just acting on **your local version of repo**.

Status and Diff

- To view the **status** of your files in the **working directory** and **staging area**:

```
$ git status
```

or

```
$ git status -s
```

(`-s` shows a short one line version)

- To see **difference** between your **working directory** and the **staging area** (This shows what is modified but unstaged):

```
$ git diff
```

- To see difference between the **staging area** and your **local copy of the repo** (This shows staged changes): (`--staged is synonymous`)

```
$ git diff --cached
```

After editing a file…

```
$emacs rea.txt
$ git status
# On branch main
# 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:   rea.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
$ git status -s
 M rea.txt                                  ← Note: M is in second column = “working  tree”
$ git diff                          ← Shows modifications that have not been staged.
diff --git a/rea.txt b/rea.txt
index 66b293d..90b65fd 100644
--- a/rea.txt
+++ b/rea.txt
@@ -1,2 +1,4 @@
 Here is rea's file.
+
+One new line added.
$ git diff --cached                 ← Shows nothing, no modifications have been staged yet.
$
```

After adding file to staging area…

```
$ git add rea.txt
$ git status
# On branch main
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   rea.txt
#
$ git status -s
M  rea.txt                          ← Note: M is in first column = “staging area”
$ git diff                          ← Note: Shows nothing, no modifications that have not been staged.
$ git diff --cached                 ← Note: Shows staged modifications.
diff --git a/rea.txt b/rea.txt
index 66b293d..90b65fd 100644
--- a/rea.txt
+++ b/rea.txt
@@ -1,2 +1,4 @@
 Here is rea's file.
+
+One new line added.
```

Uses of git diff

To see the difference between:

- Your working copy and staging area:

  $ git diff

- Staging area and the latest commit:

  $ git diff --staged

- Your working copy and commit 4ac0a6733:

  $ git diff 4ac0a6733

- Commit 4ac0a6733 and the latest commit:

  $ git diff 4ac0a6733 HEAD

- Commit 4ac0a6733 and commit 826793951

  $ git diff 4ac0a6733 826793951

Viewing logs

To see a log of all changes in your **local repo**:

- `$ git log` or
- `$ git log --oneline` (to show a shorter version)

```
1677b2d Edited first line of readme
258efa7 Added line to readme
0e52da7 Initial commit
```

- `git log -5` (to show only the 5 most recent updates, etc.)
- `git log --pretty=format:"%h%x09%an%x09%ad%x09%s"` (hash, author, date and message)

Note: changes will be listed by commitID #, (SHA-1 hash)

Note: changes made to the remote repo before the last time you cloned/pulled from it will also be included here

Pulling and Pushing

Good practice:

1. **Add** and **Commit** your changes to your local repo
2. **Pull** from remote repo to get most recent changes (fix conflicts if necessary, add and commit them to your local repo)
3. **Push** your changes to the remote repo

To fetch the most recent updates from the remote repo into your local repo, and put them into your working directory:

```
$ git pull origin main
```

To push your changes from your local repo to the remote repo:

```
$ git push origin main
```

Notes: `origin` = an alias for the URL you cloned from

`main` = the remote branch you are pulling from/pushing to,
(the local branch you are pulling to/pushing from is your current branch)

Avoiding Common Problems

- Do not edit the repository (the .git directory) manually. It wasn't designed for modifications by humans.
- Try not to make many drastic changes at once. Instead, make multiple commits, each of which has a single logical purpose. This will minimize merge conflicts. This is good coding practice in general.
- Always `git pull` before editing a file. It's easy to forget this. If you forget, you may end up editing an outdated version, which can cause nasty merge conflicts.
- Don't forget `git push` after you have made and committed changes. They are not copied to the remote repository until you do a push.

What is with "origin"? "main"?

- A working copy / local repo may be associated with:
  - No remote repo, or
  - **One remote repo** (UVic SENG 265) or
  - Several remote repos.
- A working copy / local repo may have:
  - No code branches tracked by git, or
  - **One code branch** (UVic SENG 265) tracked by git, or
  - Several code branches tracked by git.
- By git convention, the default **remote repo** is named **origin**.
- By convention, the **main branch** of code development is named **main** (i.e., it is the "master" or "main branch" of code development)

What is with "origin"? "main"?

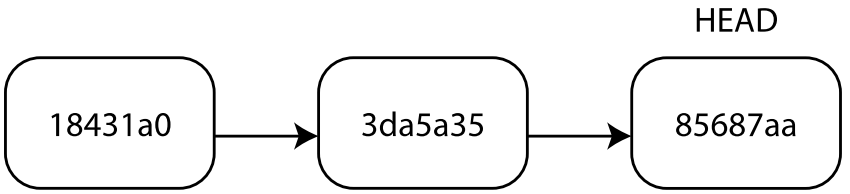
```
$ pwd
/home/stewie/calc

$ git remote -v
origin      ssh://stewie@git.example.com/repo/calc (fetch)
origin      ssh://stewie@git.example.com/repo/calc (push)

$ git branch -v
* main b5b22e2 Bevels now in place
```

- When using "git pull" and "git push" for repos already cloned:
  - origin & main are **usually** default values
  - "git pull" == "git pull origin main"
  - "git push" == "git push origin main"
- Note: We'll discuss branching workflows later in the term

What is with "origin"? "main"?



- This diagram we saw earlier of our commits / snapshot is an example of a **branch**
- The git convention is that every repo has **at least one branch** which is the main branch
  - Usually referred to as the **main** branch.
- Above is shown main branch from slide 24 (i.e., project with "calc", "a2" directories and their subdirectories/files).

A "gotcha" with git add

- This is a bit more subtle in git
- **add** results in a file or directory being staged for commit
- When **commit** is performed, changes to staged files are stored into the local repo
- Note, however:
  - Every file or directory in the project that is to be tracked by git needs to be added **at least once in the project's lifetime**
  - Also: **add** gives us fine-grained control as to what needs to be in a specific commit's snapshot
  - Sometimes, though, we just want all of the changed files to be staged and committed without having to use **add**
  - **git commit –a –m "message"**

Never store generated files in the repository.

For example, if your project includes C source code, you would store the .c and .h files. You would not store the .o or executables.

If your project includes Java source code, you would store the .java file, but you wouldn't store the generated .class or

Use .gitignore

- This text file needs to be committed to the project
- Normally stored in the top-level of the working directory
- Each line in the file is used as a pattern
- Possible entries:

```
*.pyc
*.o
DS_STORE
*.class
```