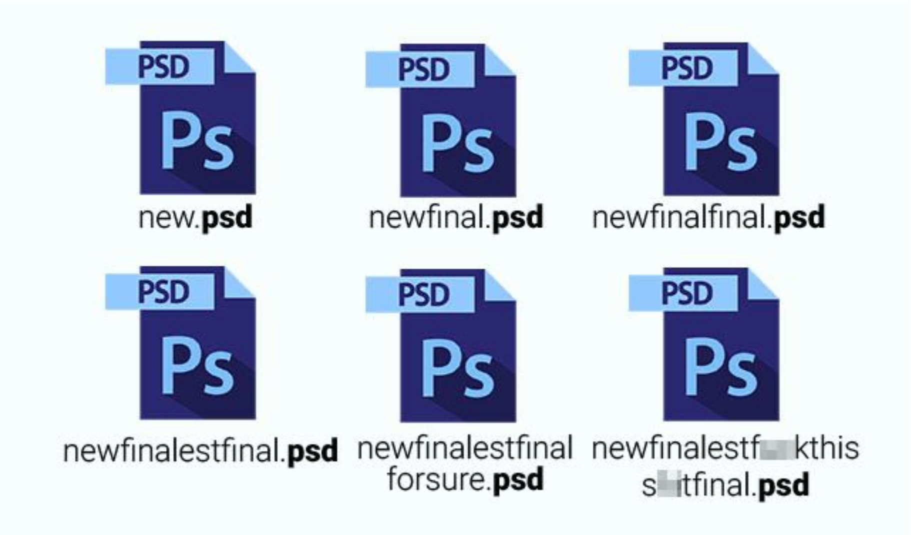
Introduction to Git

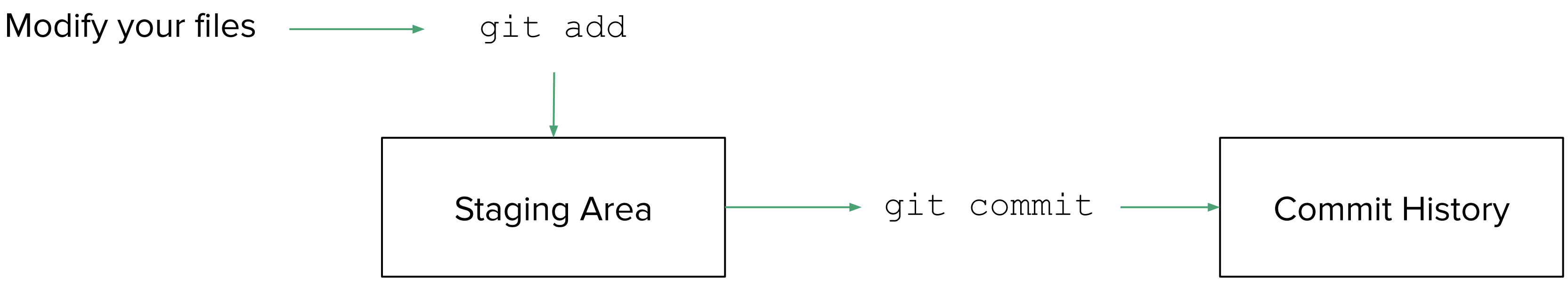
Jeriel Ng (@jerielng)
Mobile Software Developer at NCR
Auburn University 2019

What is Git?

- Version control system
- Developed by Linus Torvalds (Linux)
- Why is version control important?
    - Logged history of changes
    - Collaborative codebase


PSD
Ps
new.psd
newfinal.psd
newfinalfinal.psd
newfinalestfinal.psd
newfinalestfinal forsure.psd
newfinalestf[redacted]kthis s[redacted]tfinal.psd

Commit Process



Modify your files

`git add`

Staging Area

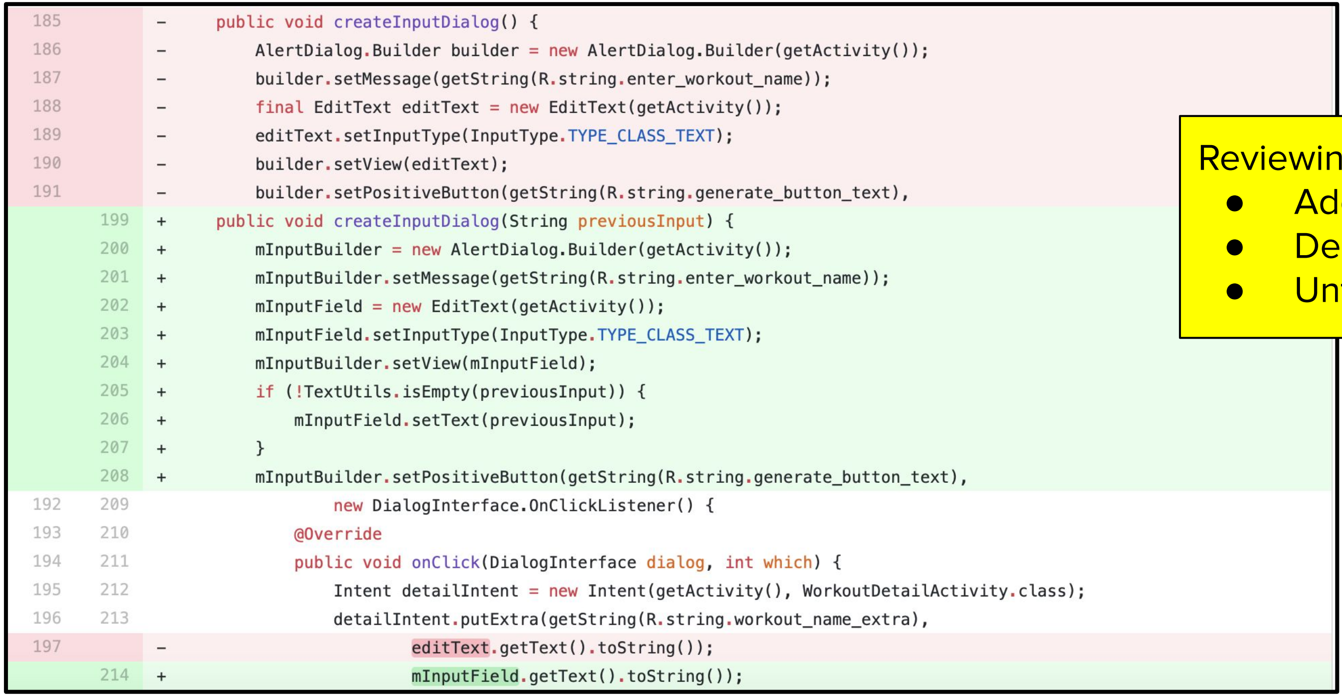
`git commit`

Commit History

What is a commit?

A single point of time in your Git history that contains a log of changes

Commit History: Reviewing Your Changes



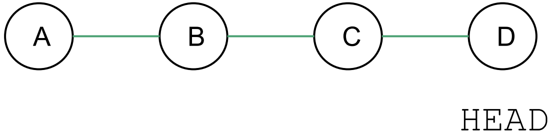
```diff
-    public void createInputDialog() {
-        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
-        builder.setMessage(getString(R.string.enter_workout_name));
-        final EditText editText = new EditText(getActivity());
-        editText.setInputType(InputType.TYPE_CLASS_TEXT);
-        builder.setView(editText);
-        builder.setPositiveButton(getString(R.string.generate_button_text),
+    public void createInputDialog(String previousInput) {
+        mInputBuilder = new AlertDialog.Builder(getActivity());
+        mInputBuilder.setMessage(getString(R.string.enter_workout_name));
+        mInputField = new EditText(getActivity());
+        mInputField.setInputType(InputType.TYPE_CLASS_TEXT);
+        mInputBuilder.setView(mInputField);
+        if (!TextUtils.isEmpty(previousInput)) {
+            mInputField.setText(previousInput);
+        }
+        mInputBuilder.setPositiveButton(getString(R.string.generate_button_text),
                 new DialogInterface.OnClickListener() {
             @Override
             public void onClick(DialogInterface dialog, int which) {
                 Intent detailIntent = new Intent(getActivity(), WorkoutDetailActivity.class);
                 detailIntent.putExtra(getString(R.string.workout_name_extra),
-                        editText.getText().toString());
+                        mInputField.getText().toString());
```

Reviewing your diff:

- Additions - green lines
- Deletions - red lines
- Untouched - white lines

Commit History: Undoing Your Changes

First off,
what is HEAD?



HEAD is the most recent commit of the branch you’re currently on

```
git reset --hard
```

Completely discard changes, whether staged or not

Default

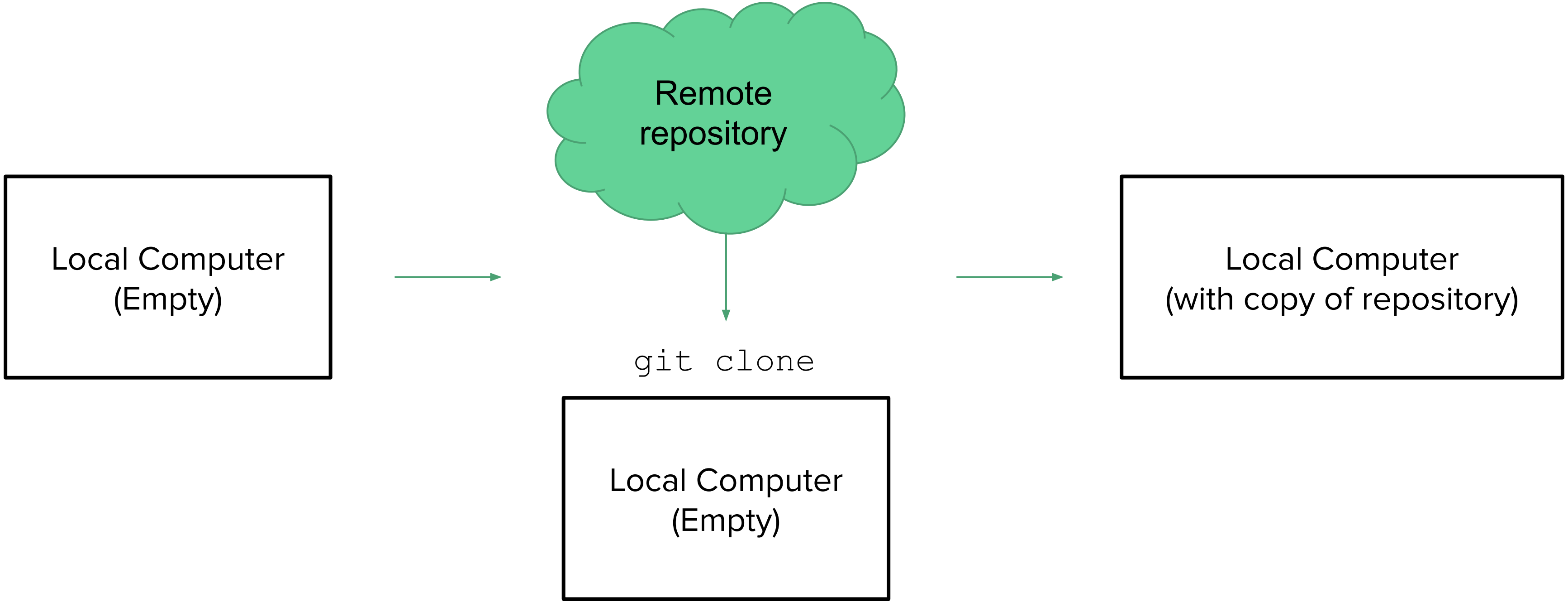
```
git reset --mixed
```

Removes files from staging area, but keeps changes

```
git reset --soft
```

Moves HEAD pointer, but keeps changes **in** the staging area

Working with Remotes: Setup



Remote repository

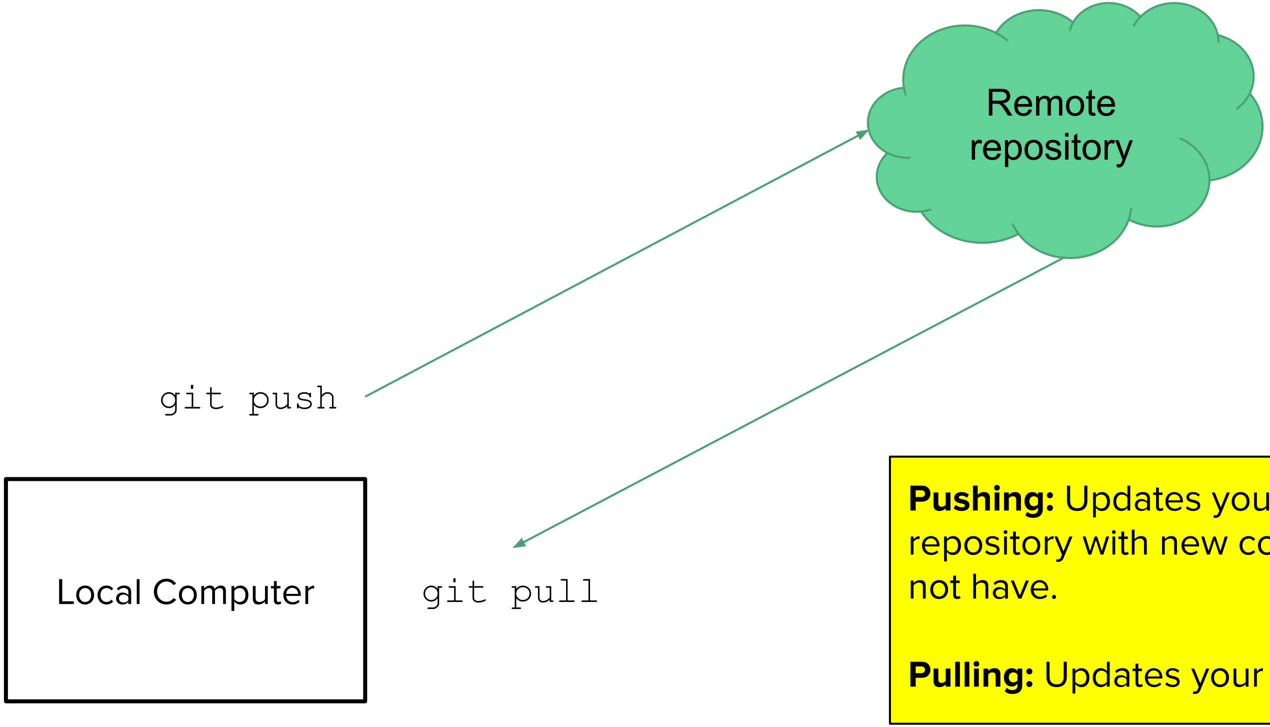
Local Computer (Empty)

git clone

Local Computer (Empty)

Local Computer (with copy of repository)

Working with Remotes: Updating



Remote repository

`git push`

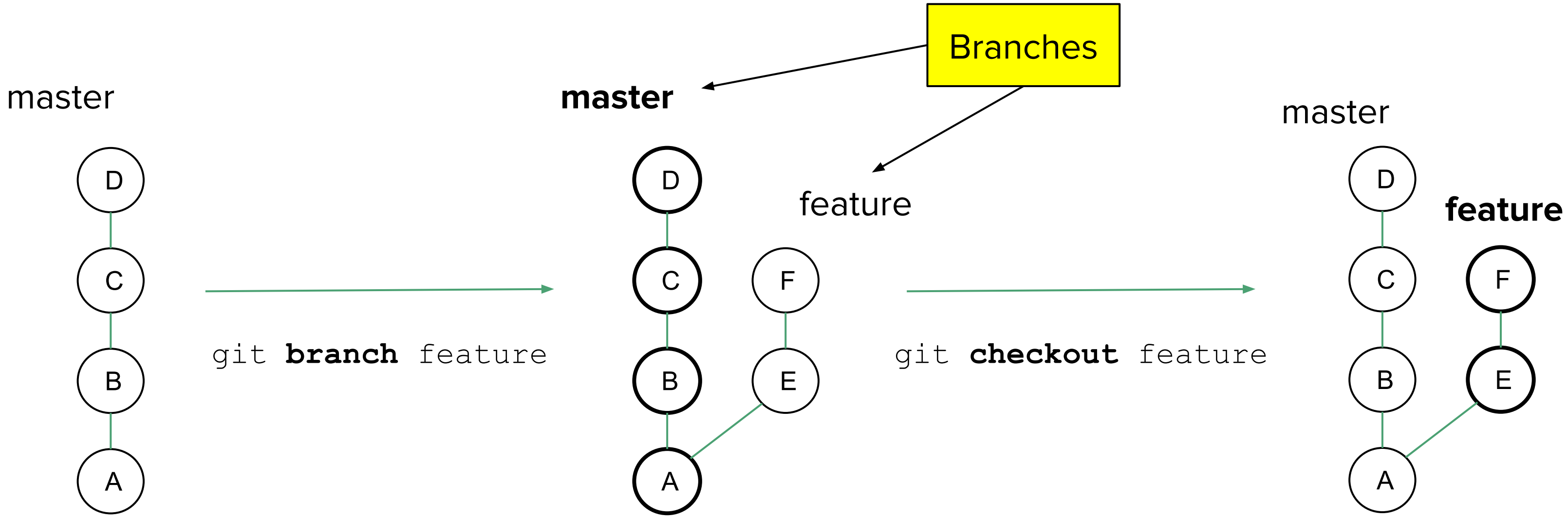
Local Computer

`git pull`

Pushing: Updates your remote repository with new commits it does not have.

Pulling: Updates your local repository.

Alternate Timelines: Branches & Forks

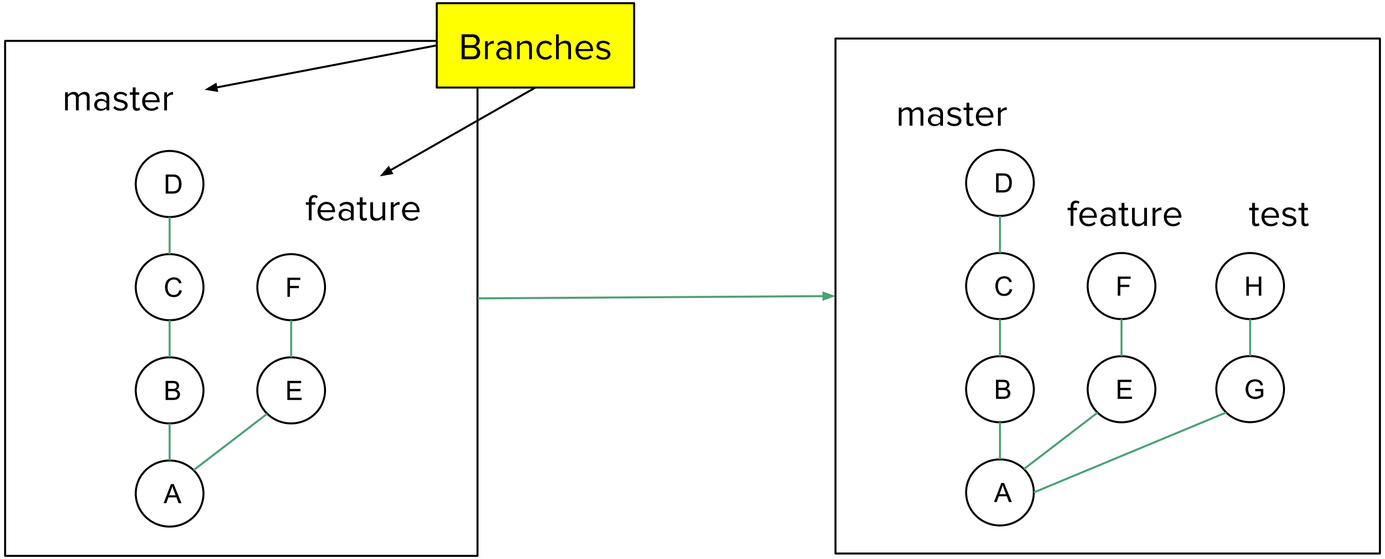


git **branch** feature

git **checkout** feature

You can also accomplish this with a single line:
git **checkout -b** feature

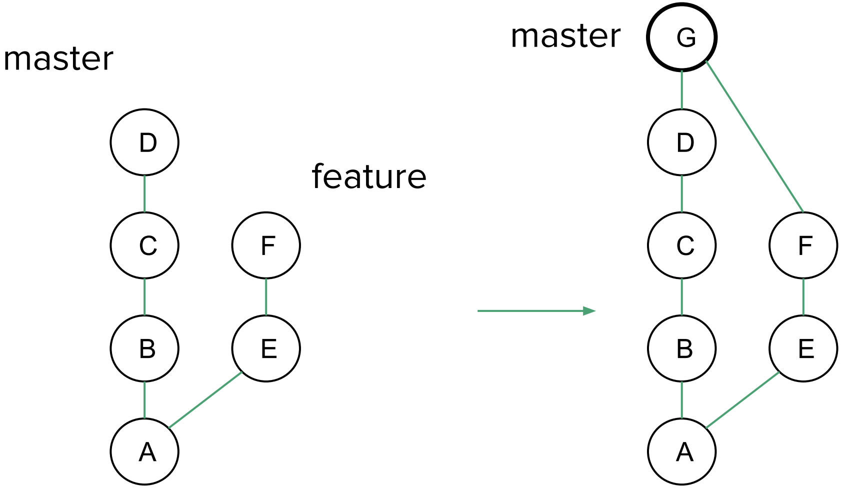
Alternate Timelines: Branches & Forks



`awesome_project`

Fork of `awesome_project`

Merging vs. Rebasing

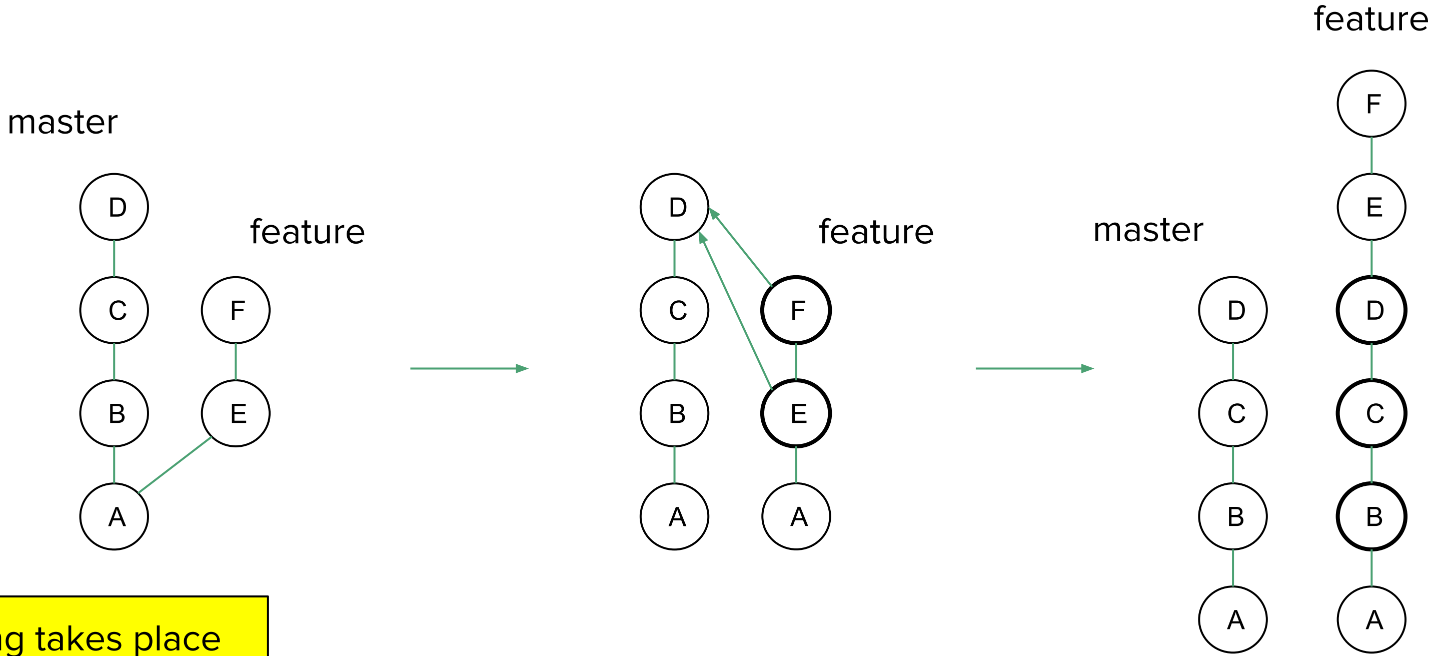


G is a newly created "merge commit" to combine the histories of both master and feature

```
(master): git merge feature
```

Notice merging takes place from the **master** branch, not the **feature** branch

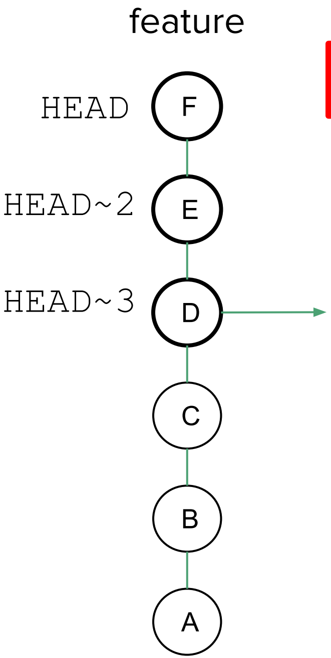
Merging vs. **Rebasing**

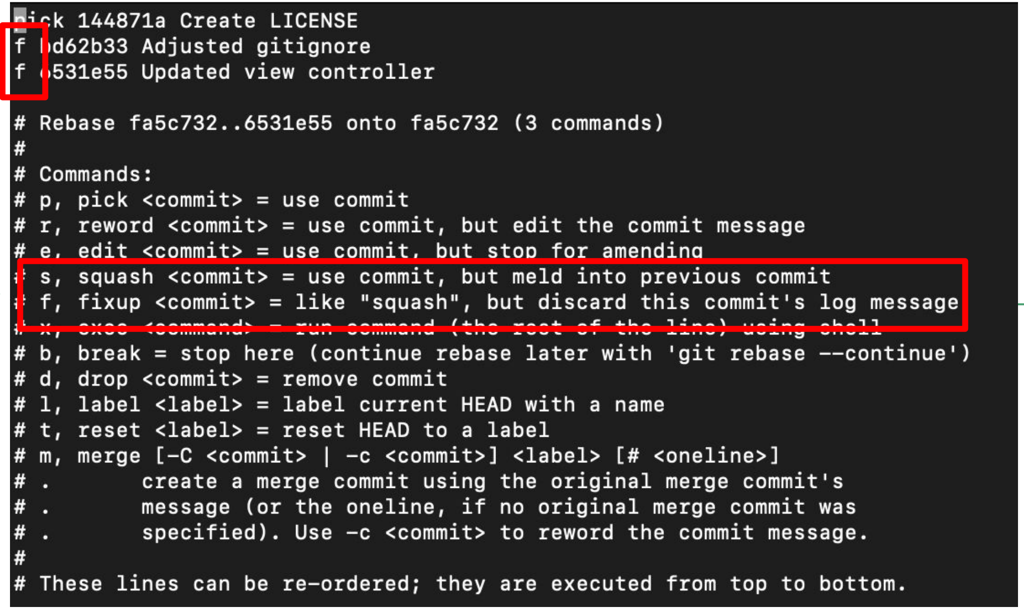


Notice rebasing takes place from the **feature** branch, not the **master** branch

```
(feature): git rebase master
```

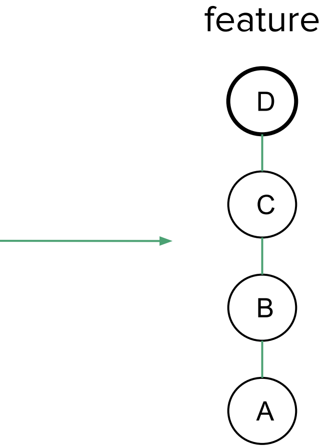
Squashing Your Commits





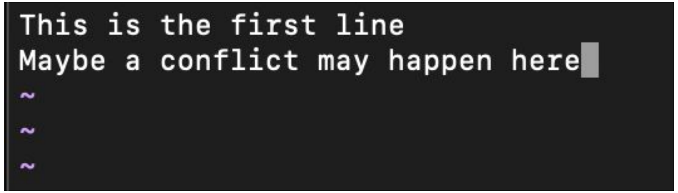
```
pick 144871a Create LICENSE
f bd62b33 Adjusted gitignore
f 6531e55 Updated view controller

# Rebase fa5c732..6531e55 onto fa5c732 (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
```



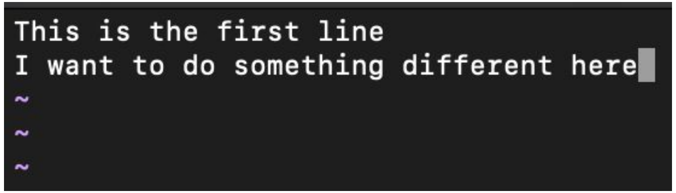
`(feature): git rebase -i HEAD~3`

Merge Conflicts & How to Resolve Them



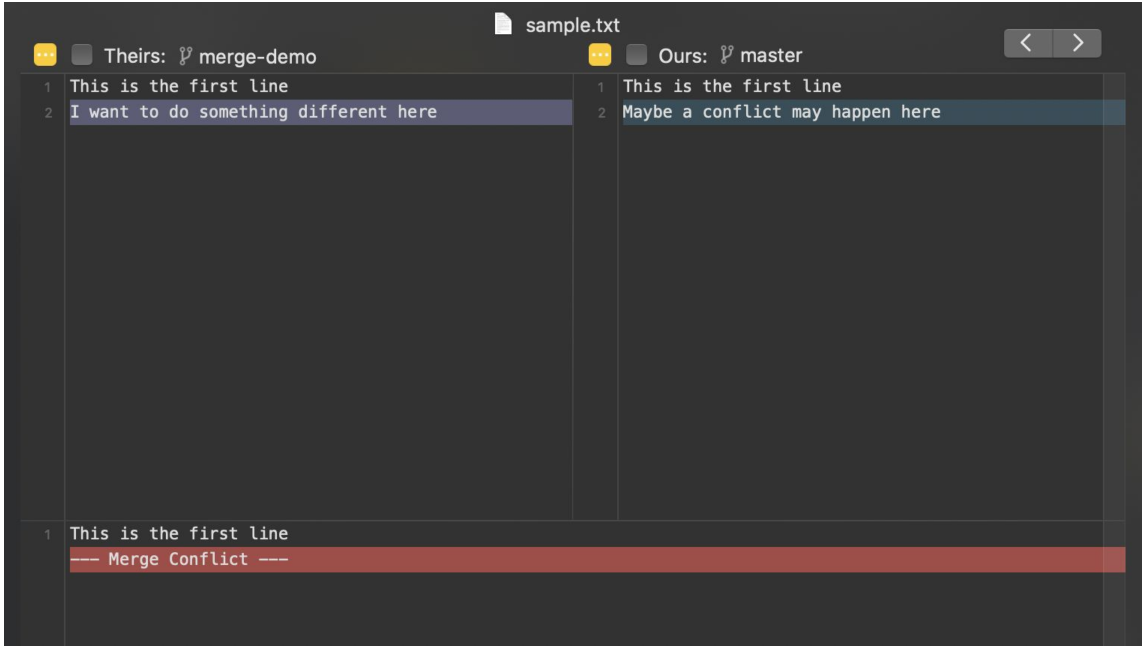
```
This is the first line
Maybe a conflict may happen here
~
~
~
```

Coder A’s changes



```
This is the first line
I want to do something different here
~
~
~
```

Coder B’s changes



```
This is the first line
I want to do something different here
```

```
This is the first line
Maybe a conflict may happen here
```

```
This is the first line
--- Merge Conflict ---
```

Merge Conflicts & How to Resolve Them

- Best approach: **Use a text editor or GUI tool** (Especially for big files)
  - Visual Studio Code
  - GitKraken
  - Fork
  - SourceTree
- Have an understanding of **who** changed **what** and **why**
- Run your code afterwards to make sure it compiles

What is GitHub? I’ve heard of that

Built on top of Git

- GUI-based platforms
- Features:
  - Repository Hosting
  - Pull Requests
  - CI/CD Integration

Further Resources

- Git Reference Manual
- Interactive Tutorials on Git
- GitHub Learning Lab
- Desktop GUI tools for Git
  - Fork
  - GitKraken
  - Sourcetree

Questions