```
# This is a synopsis of what we’ll be doing in class.  I’ll demonstrate,
# while you follow along.
#
# Assuming:
#     1) You’ve sent me your GitHub username
#     2) You’ve accepted the invitation  to join the
#        GoucherCollegeCS411Spring2022 organization sent to the email
#        associated with your GitHub account
#     3) I’ve had time to add you to the cs411 team

# From Canvas, follow the link to create the GitHub individual work
# repository.

# Open a terminal window.

# Let’s be very sure you’re in your home directory.

cd

# Don’t ignore the period at the end of the command line.

cp -i ~kelliher/pub/cs411/.gitconfig .

# Or substitute your favorite text editor.  Change name, email, and,
# perhaps, editor.  Save and exit.

geany .gitconfig

# Back in your home directory...

mkdir Cs411

cd Cs411

# Using a web browser, log into GitHub and go to the class organization.

# The following command will fail unless the three assumptions above all hold.
# You can’t move on until this git clone succeeds.  Replace ... with the
# URL for the Cs411GitHubRepo in the class organization.
#
# You’ll be asked for your GitHub username and a "password."  Your GitHub
# password WILL NOT WORK!!!  Go to the settings page for your account, then
# Developer settings, then Personal access tokens, and generate a new
# token.  Use this token for your "password" here.

git clone ...

cd Cs411GitHubRepo

# Rename the original remote repository.  You’ll get new assignments from
# this remote repository, named upstream.

git remote rename origin upstream

# Replace ... here with the URL of your work- repository in the class
# organization.  The origin remote repository is where you publish your
# work.

git remote add origin ...

# The ’-u origin main’ here sets up your local main branch to "track" the
# origin remote’s main branch.
```

```
git push -u origin main

# At this point, you're ready to start working.  Start with hello.c in the
# Practice directory in the repo.  Open it and follow the instructions.
#
# As you make and save changes, run

git status

# from time-to-time so that you can see suggestions for changed files
# which should be "staged" for commit.  hello.c will be one.  Here's how to
# stage it:

git add hello.c

# Not all files should be staged.  Executable files are a good example.
# The hidden file .gitignore is used to inform git which files should be
# ignore.  Create a .gitignore file and add hello and practice to it, one
# file name per line.  Save the file and add it to be staged.
#
# Sets of related changes are collected into commits.  Once you've staged
# all your changes for a commit, run this command:

git commit

# An editor will pop up so that you can write a commit message.  The
# message should be written in the present tense and summarize the changes
# being made in the commit.  The commit will happen once your save the
# commit message and exit the editor.
#
# The convention is that commits happen fairly frequently.
#
# It's also the convention to not perform development on the main branch.
# We'll look into this later.
#
# You push your commits to publish them.  It's a good idea to synchronize
# with the remote before publishing:

git pull

# This performs a fetch and a merge.  Hopefully, you won't have a merge
# conflict, but we'll see how to handle this later.
#
# Now, publish your commits:

git push

# Note that we don't need to use the '-u origin main' option again.
```