**How Often to Commit**

Since you can choose when to make a commit, you might be wondering how often to commit your changes. It's usually a good idea to keep commits small. As the diff between two versions gets bigger, it gets harder to understand and less useful. However, you don’t want to make your commits too small either. If you always save a commit every time you change a line of code, your history will be harder to read since it will have a huge number of commits over a short time period.

A good rule of thumb is to make *one commit per logical change*. For example, if you fixed a typo, then fixed a bug in a separate part of the file, you should use one commit for each change since they are logically separate. If you do this, each commit will have one purpose that can be easily understood. Git allows you to write a short message explaining what was changed in each commit, and that message will be more useful if each commit has a single logical change.

**Commit Size Quiz**

To get some practice thinking about how often to commit, on the next screen, mark whether you think the following would be good commit sizes. If not, indicate whether you think this commit is too small and you’d like to wait and commit later, or whether you think it’s too big and you would have committed earlier. This is subjective, so there aren’t any definite right or wrong answers, but just choose the answer you think is best in each case.

* You commit all the changes required to add a new feature, which you’ve been working on for a week. You haven’t committed since you started working on it.
* You find three typos in your README. You fix and commit the first.
* You commit all the changes required to add a new feature, which you’ve been working on for an hour.
* You fix two small bugs in different functions and commit them both at once.
  1. Git Log ---stat to know changes in which file the changes have been in commit
  2. Git diff – to know difference in commits
  3. Git clone
  4. Git scp= secure copy

### Cloning a Repository

To clone a repository, run git clone followed by a space and the repository URL.

### Asteroids URL

Use the following url to clone the Asteroids repository: [**https://github.com/udacity/asteroids.git**](https://github.com/udacity/asteroids.git)

### Exiting git log

To stop viewing git log output, press q (which stands for quit).

### Getting Colored Output

To get colored diff output, run git config --global color.ui auto

### Copying and Pasting from the Command Line

To complete this quiz, you'll want to copy and paste some commit ids.

**Windows**  
To copy and paste within Git Bash, follow the instructions on ctrl+shift+ins

### Using git log and git diff

As a reminder, running git log will show a list of the recent commits with information about them, including commit IDs. Running git diff followed by two commit IDs will compare the two versions of the code in those commits.\

Entering commit IDs

If it is easier, you may enter the first four or more characters of the commit ID rather than pasting the entire ID.

If you’re having some trouble checking out a previous version of a commit, check to see if your error is covered here. If not, please post your question on the forums. You get the following error: error: Your local changes to the following files would be overwritten by checkout: game.js Please, commit your changes or stash them before you can switch branches. Aborting This message occurs if you’ve made changes to any of the files. Since these changes aren’t committed yet, if git checkout out a previous commit, they would be lost. You can see what changes you’ve made using the command git diff with no arguments. (If you don’t give git diff any arguments, it compares the current state of your files to the most recent commit.) If you don’t care about the changes, you can use git reset ­­hardto get rid of them, but be careful! This command cannot be undone. If you want to keep the changes, you can commit them, but first you’ll need to let git know your name and email address by entering the commands git config ­­global user.name "Your name"and git config ­­global user.email "youremail@domain.com". Then you can commit your changes using git commit ­a ­m "Commit message"where the commit message can be anything you want. The ­a flag means to commit all the changes you’ve made. You’ll see in Lesson 2 how to choose what to commit. If you forget to use the ­m flag, git will open a text editor for you to type a commit message. This editor will be vim, which can be confusing to use if you haven’t used it before. and if you haven’t used it before, it can be confusing. To quit vim, press the escape key, then type :q!You should see a message that says Aborting commit due to empty commit message. Now go ahead and try committing again, but this time remember the ­m flag. When you make the commit, you should get a message like [master 91ef3c2] Commit message.The string of letters and numbers is the beginning of your commit’s ID. You can use this to refer to your commit. For example, you can use git checkout 91ef3c2to get to this commit again. You can’t find a commit via git log git log only shows commits that were made before the commit you have currently checked out. If you want to see all the commits we saw at the beginning of the lesson, try using the command git checkout abcdef, which is the most recent commit. If you’ve made your own commit and can’t remember its commit ID, try using the command git reflog, which shows you all the commits that have been checked out recently. The strings of numbers and letters on the left of the reflog output are the beginnings of commit IDs, and you can use this prefix to refer to a commit just like you use the full ID.