LESSONS

1. Workshop overview & setup
2. What is version control?
3. Basic git commands
4. Undoing things
5. Navigate GitHub repos
6. Fork a GitHub repo
7. Create pull requests

Lesson 4. Introduction to undoing things in git

Intro version control git

Max Joseph, Leah Wasser

🎓 Learning objectives

At the end of this activity, you will be able to:

- Undo changes before they have been staged
- Unstage modified files after they have been staged
- Revert back to a previous commit if unwanted changes have been committed

☑ What you need

- A GitHub user account
- A terminal running bash, and
- Git installed and configured on your computer.

Follow the setup instructions here:

- Setup instructions

This lesson describes how to undo changes:

1. before they've been staged (you haven't used `git add` yet to add or stage them),
2. after they've been staged with `git add`, and
3. after they've been committed to git.

Undoing unstaged changes

If a file has been changed, but these changes have not yet been staged with `git add`, then the changes can be undone using `git checkout`. The instructions for using git checkout to undo changes are described in the output of `git status`.

Let's look at an example. First, let's modify the readme file by adding some text to it at the command line.

```
# append "Some random text" to the README file in shell
echo 'Some random text' >> README.md
```

Next, type git status to see how that change impacted git

```
$ git status
```

In the example above, git status was run in the command line after a file was edited. When you run git status, git will first provide the following output:

```
On branch master
Your branch is up-to-date with 'origin/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:   README.md

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

The output from `git status` tells you that you can use `git checkout -- <file>` to discard changes to that file in your repo. So, if you don't like the changes made to the README.md file, you can revert back to the last committed version using:

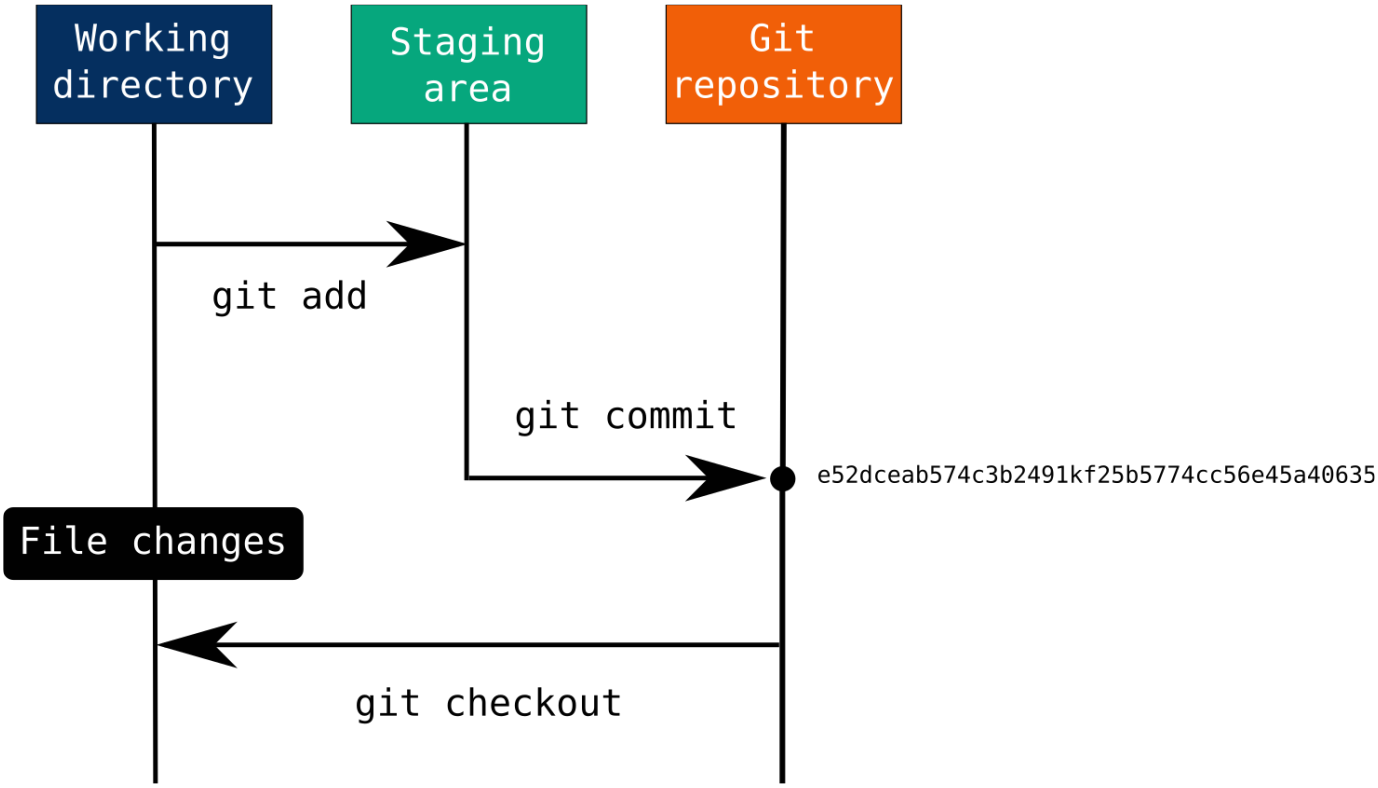
```
git checkout -- README.md

git status
```

Which returns:

```
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

Now, the contents of your README.md file has been reverted to the last saved or committed version and you've discarded the most recent changes.



Git checkout can undo unstaged changes by pulling the previous commit's version of a file from repository's history. Source: Maxwell Joseph, adapted from Pro Git by Chacon and Straub (2014).

Challenge - use git checkout to undo changes

Let's see how git checkout works.

1. Make a few text changes to your `README.md` file. You can make these changes in shell using the example below OR your favorite text editor. Save your changes (if you're in a text editor).
2. Now go to bash if you aren't already there. Run `git status`
3. Undo the changes that you made using `git checkout`

Unstaging staged changes

Remember that once you add a set of changes to git using `git add`, the file is then staged. If a file has been changed and then staged via `git add`, then you use `git reset` to pull the most recently committed version of the file and undo the changes that you've made.

Fortunately, the output of `git status` gives us a hint for how to undo our staged changes:

```
# modify the README file
echo 'Some more changes' >> README.md

git add README.md
git status
```

```
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
```

You use `git reset HEAD <file>` to unstage our changes. `HEAD` refers to the most recently committed version of the file:

```
git reset HEAD README.md
```

```
Unstaged changes after reset:
M       README.md
```

★ **Data tip:** HEAD refers to the most recent version of your file. You can also revert to an older version using HEAD~1, HEAD~2 etc. Read more about this on the Software Carpentry git lessons website.

When you use git reset, your changes still exist in the file, but the file has been unstaged (the changes are not added to git, yet).

```
git status
```

```
On branch master
Your branch is up-to-date with 'origin/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:   README.md

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

Now that you have changes that are not staged, you can use `git checkout` to undo those modifications. `Git reset` is essentially the opposite of the command `git add`. It undoes the `add`.

Challenge - use git reset then checkout to undo changes

Practice using git reset and git checkout.

1. Make a few text changes to your `README.md` file. You can make these changes in shell using the example below OR your favorite text editor. Save your changes (if you're in a text editor).
2. Now go to bash if you aren't already there. Run `git status`
3. Use `git add` to stage your changes to the README.md file.
4. Undo the commit that you made using `git reset`. Then revert back to the previously committed version using `git reset`.

Undoing a commit

If you have modified, added and committed changes to a file, and want to undo those changes, then you can again use `git reset HEAD~` to undo your commit. Similar to the previous example, when you use `git reset` the modifications will be unstaged.

```
# create a sensitive file
echo '123-45-6789' >> social-security.txt

# add it
git add --all

# commit
git commit -m 'Accidentally including my social security number in my file'

git status
```

```
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
```

Now you can undo this commit with `git reset HEAD~`:

```
git reset HEAD~

git status
```

```
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        social-security.txt

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

Notice that now your file is no longer being tracked!

If you inspect the output of `git log`, you will notice that your previous commit is no longer part of the repository's history.

Ignore sensitive files

If you do have sensitive files in a repository that you never want to track with git, you can add those file names to a file called `.gitignore`, and git will not track them. For instance, if you have a text file that contains sensitive information such as a social security number called: social-secutity.txt that you don't want to keep track of, you can add that file to a `.gitignore` file.

The .gitignore file lives in the home directory of your repo.

```
# create a .gitignore file - only do this if one doesn't already exist
touch .gitignore
```

Now open the that file in a text editor and add the following lines below:

```
# contents of the .gitignore file
social-security.txt
```

Any files listed in this file will be ignored by git. You can also tell git to ignore entire directories.

★ **Data tip:** Learn more about using `.gitignore` files to ignore files and directories in your git repo on the Software Carpentry git lessons website.

Additional resources

- On undoing, fixing, or removing commits in Git
- Git basics - undoing things

Basic git commands Navigate GitHub repos

Tags **Reproducible science and programming:** git , version control
Updated: September 03, 2019