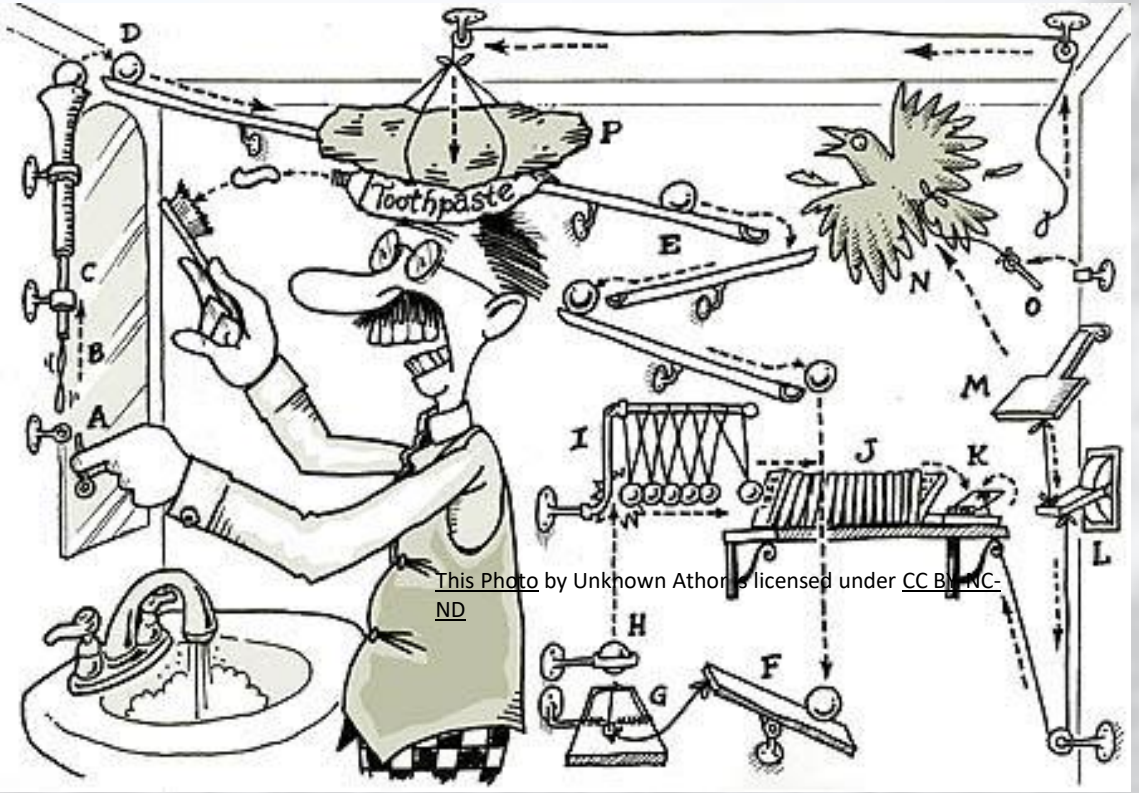
Git
Distributed Version Control

When I work alone ……

- Life is simple!
  - ❖I write my code
  - ❖I test my code
  - ❖I save my code
- And then I am done!

But when I collaborate….

- Life gets complicated
  - ❖We are all writing code for the same project
  - ❖I don’t want anyone to use what I have done until I know it works
  - ❖I need their changes and additions …. But only after they are known working
  - ❖What if we both change the same part of the code?
  - ❖What if somebody screws up?



Working with Git

- One core concept when working with Git is a Repository (aka Repo)

- Git is an example of a *distributed* Version Control System, so there are always at least two repos involved:
  - Local
  - Remote

The local repository is on your machine

The remote repository is on a tropical island somewhere*

*OK, maybe it isn't really on a tropical island. But I like to think of my code relaxing on a beach after a hard days work!

Creating a Git Project

```
gitProject — -bash — 80×24
Bills-MacBook-Pro:projects supremecommanderstuff$ mkdir gitProject
Bills-MacBook-Pro:projects supremecommanderstuff$ cd gitProject
Bills-MacBook-Pro:gitProject supremecommanderstuff$ git init
Initialized empty Git repository in /Users/supremecommanderstuff/Desktop/Project
s/gitProject/.git/
Bills-MacBook-Pro:gitProject supremecommanderstuff$
```

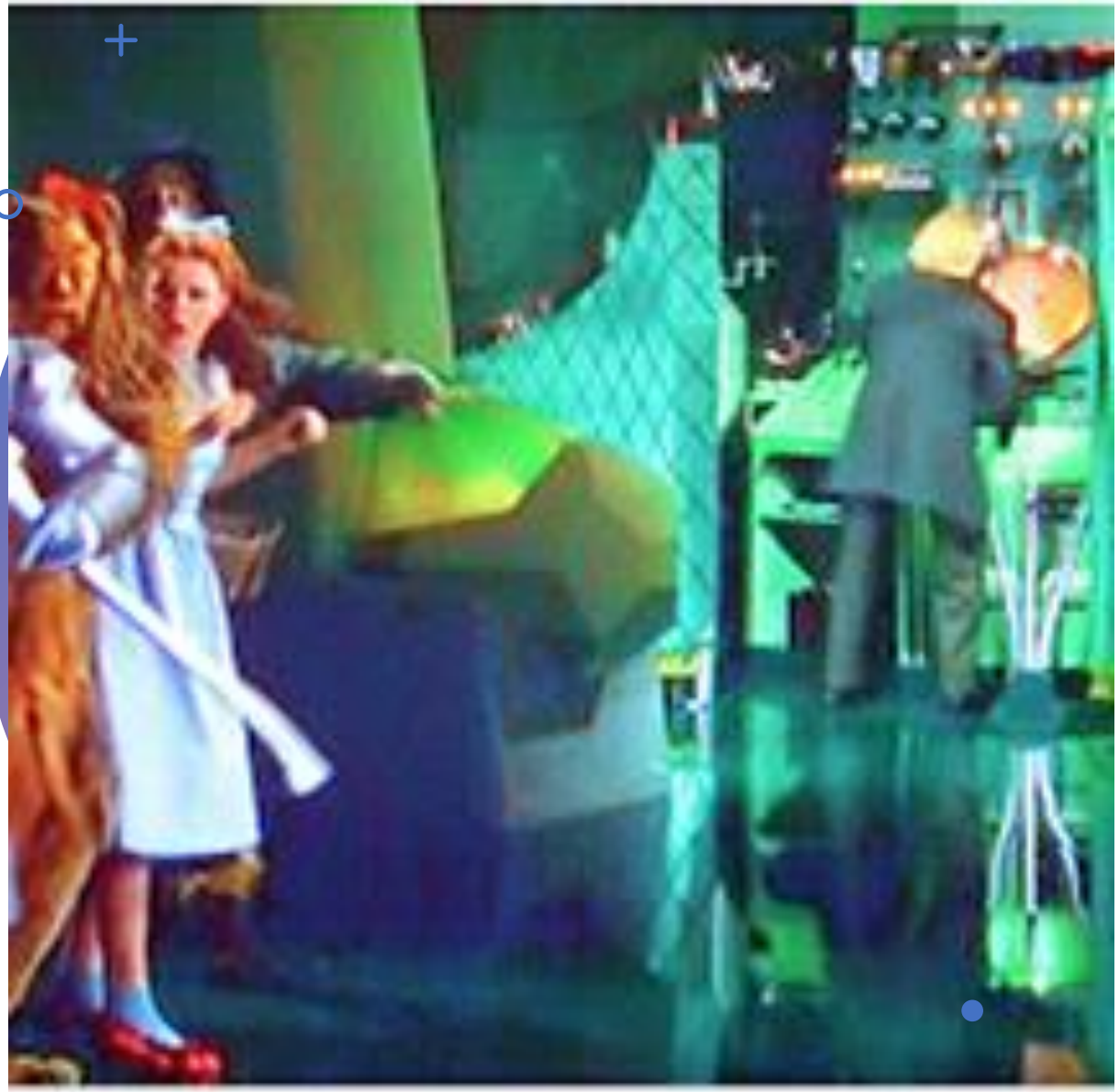
1

2

3

Steps:

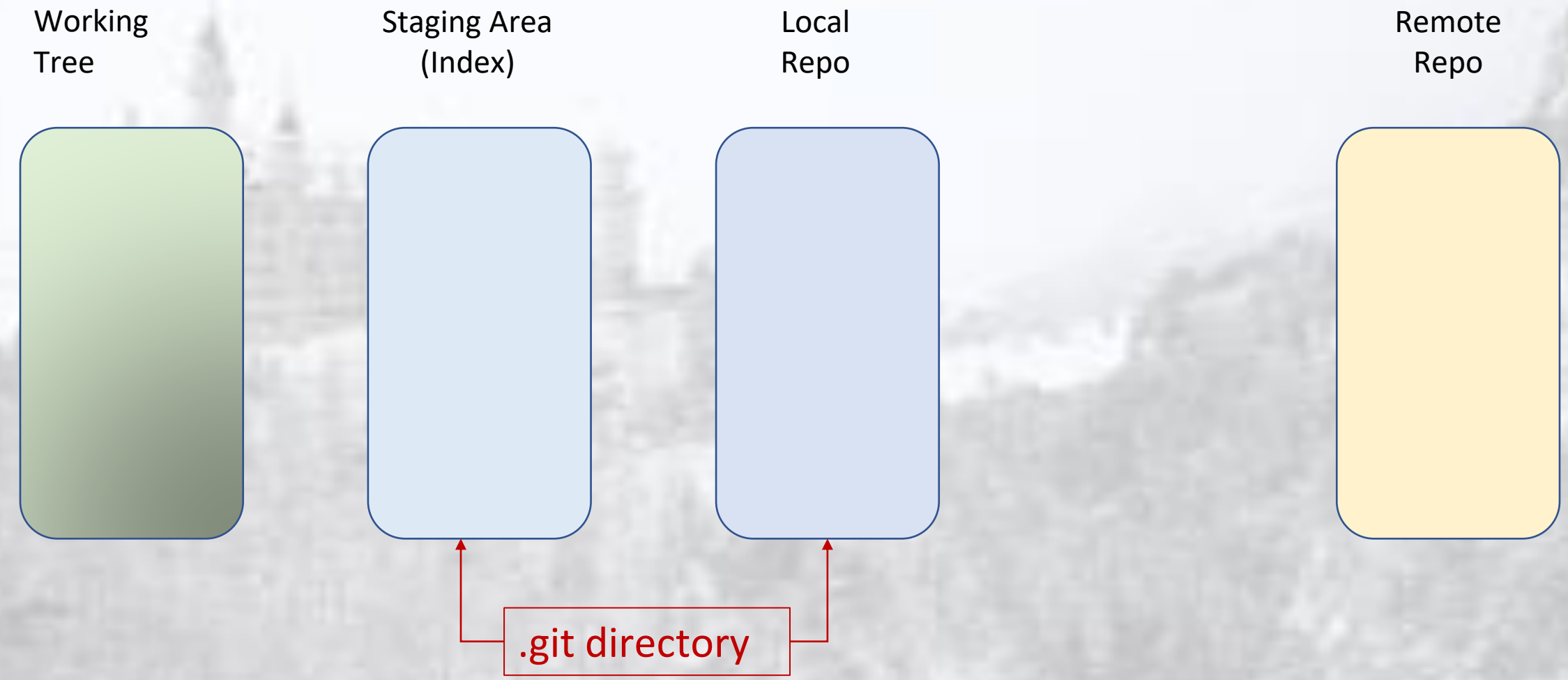
1. Create a directory for your new project (I named mine “gitProject”)
2. Navigate to that directory
3. Run $ git init



Behind the scenes …. Part 1

- The git init command tells git that this directory should be tracked by git
- Git puts a hidden file (.git) into your directory
  - All of the changes will be recorded in this directory.

Behind the scenes …. Part 2



Behind the scenes …. Part 2

Working Tree

We work here
- Edit
- Create
- Delete
- etc

Staging Area (Index)

We move finished work we want to commit to the repository here

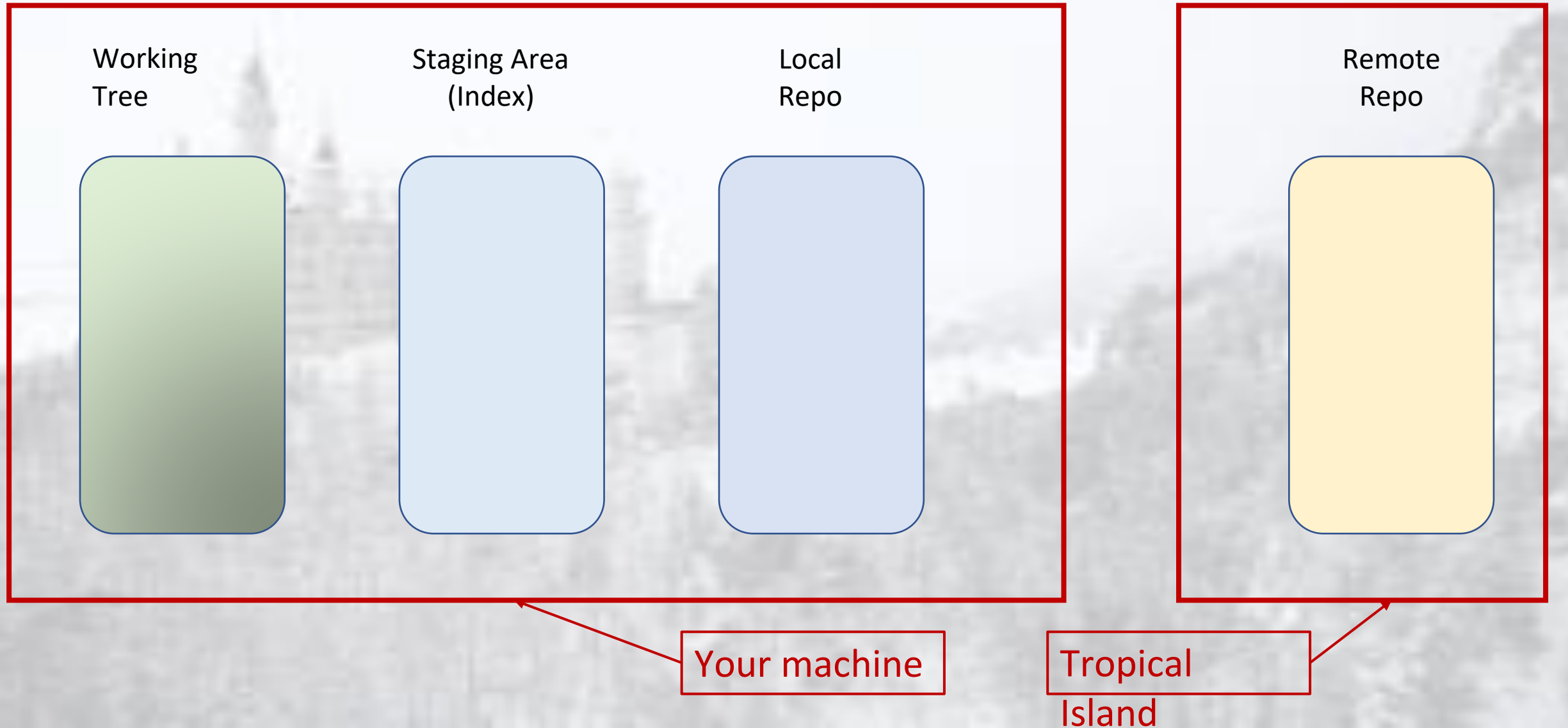
Local Repo

This contains a complete record of our commit history

Remote Repo

This contains files we want to share with others

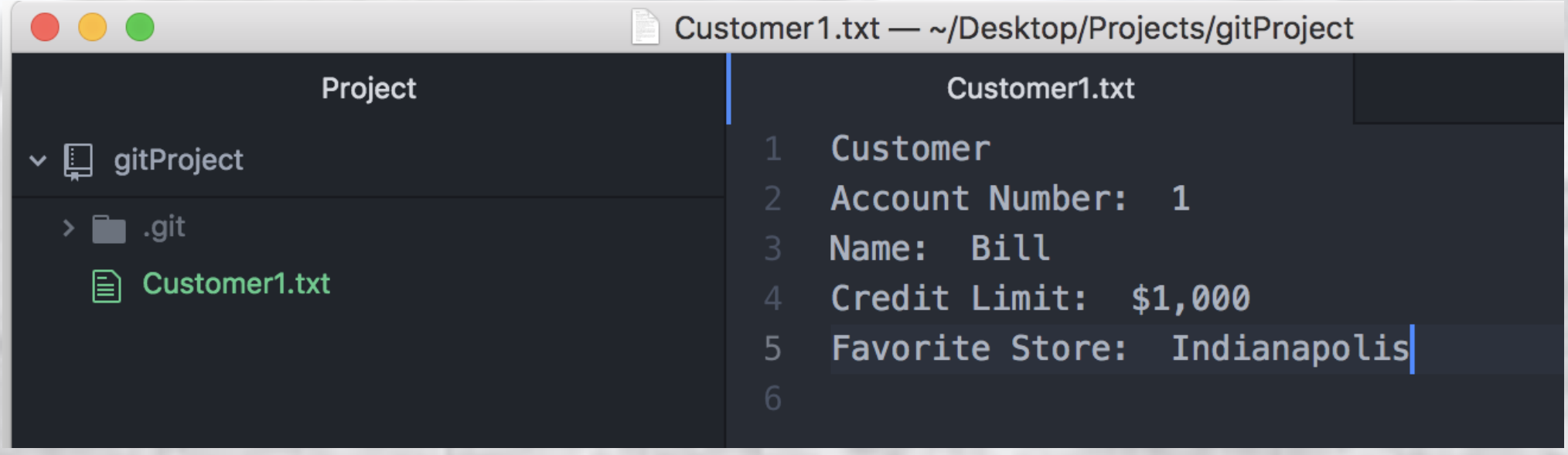
Behind the scenes …. Part 2



The Working Area

Open a text editor and create a new file for our project. (I made Customer1.txt)

Save the file



Customer1 created and Saved

Git considers Customer1 to be “Untracked”, since we haven’t previously committed it

Git Status command

Running the git status command

Git tells us we have one untracked file

Git tells us what to do next

gitProject — -bash — 80×24

```
Bills-MacBook-Pro:gitProject supremecommanderstuff$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        Customer1.txt

nothing added to commit but untracked files present (use "git add" to track)
Bills-MacBook-Pro:gitProject supremecommanderstuff$
```

Add the file to the staging area with $git add

After running $git add …… Customer1 is now a tracked file

Re-running $git status

- Git tells us that there is a new file that can be committed to the repository (ie, a file has been staged)

```
gitProject — -bash — 80×24
Bills-MacBook-Pro:gitProject supremecommanderstuff$ git add Customer1.txt
Bills-MacBook-Pro:gitProject supremecommanderstuff$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   Customer1.txt

Bills-MacBook-Pro:gitProject supremecommanderstuff$
```

It also tells us how to remove the file from the staging area if we don't want to commit it

Move the file to the local repository with $git commit

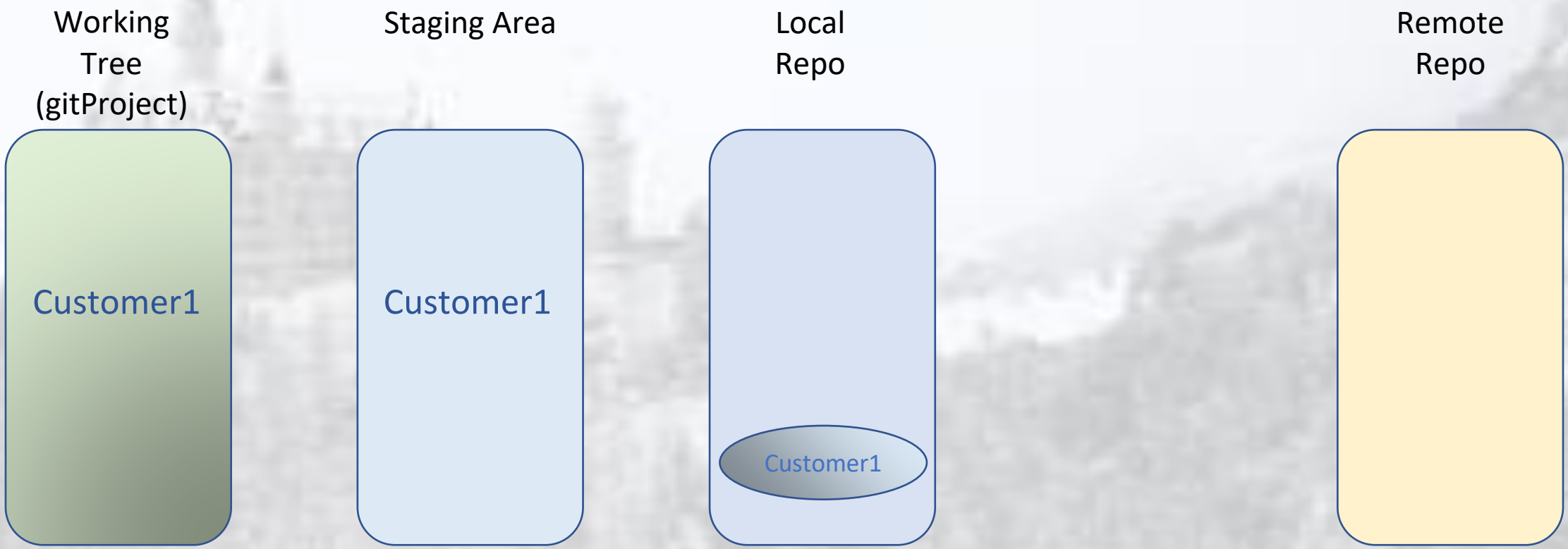
$git commit moves everything in the staging area to the local repository

The –m is for adding a message to the commit

```
Bills-MacBook-Pro:gitProject supremecommanderstuff$ git commit -m "add Customer1"
[master (root-commit) f9dde33] add Customer1
 1 file changed, 5 insertions(+)
 create mode 100644 Customer1.txt
Bills-MacBook-Pro:gitProject supremecommanderstuff$
```

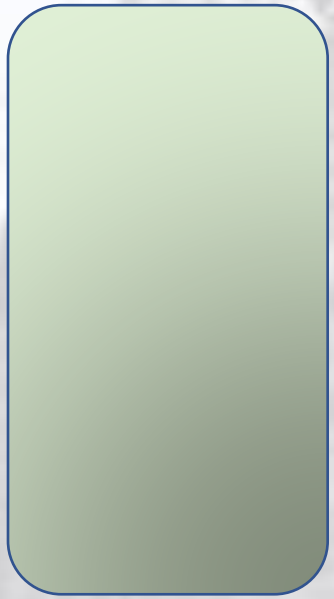
Git creates a 40 digit SHA-1 hash to identify this commit action. It displays the first 7 digits

Committing the change to the local repo with $git commit -m



Recap 1

gitProject



Step 1: We created the directory that will hold all of our files and subdirectories

Recap 2

Step 2: We told git to track this directory with $git init

Git created the .git directory, and the corresponding staging area and local repo. It also provisioned the remote repo

Recap 3

Step 3: we created a new file in our working tree (gitProject)

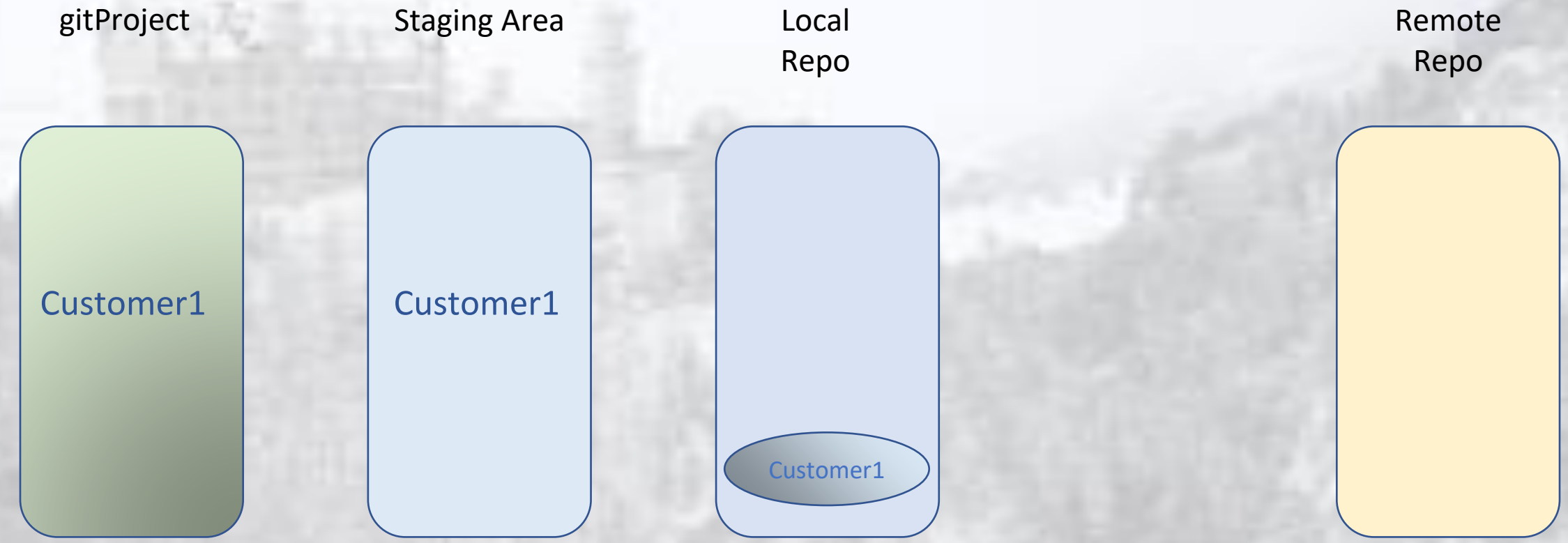
Recap 4

Step 4: We added the file to the staging area in a two step process

- $git status to see if there were any untracked changes
- $git add Customer1.txt to stage the file

Recap 5

Step 5: we ran
$git commit
to commit all of the files in the staging area to the local repo. The local repo created a record of this commitment



Question …..

What would you expect to happen if we ran
$git status
after we committed our changes?

gitProject — -bash — 83×24

```
Bills-MacBook-Pro:gitProject supremecommanderstuff$ git status
On branch master
nothing to commit, working tree clean
Bills-MacBook-Pro:gitProject supremecommanderstuff$
```

$git log

$git log
is used to display the history of commits to our local repository

So far, there has only been one commit made

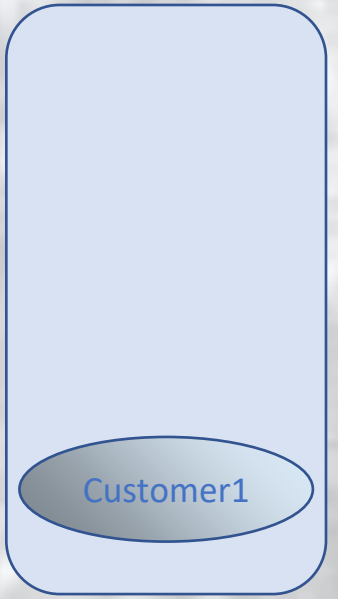
```
Bills-MacBook-Pro:gitProject supremecommanderstuff$ git log
commit f9dde332f9718eb33eac4147daffef083c4cc70b (HEAD -> master)
Author: Bill Fairfield <bill@fairfieldco.com>
Date:   Sat Jun 9 14:10:36 2018 -0400

    add Customer1
Bills-MacBook-Pro:gitProject supremecommanderstuff$
```

40 digit hash

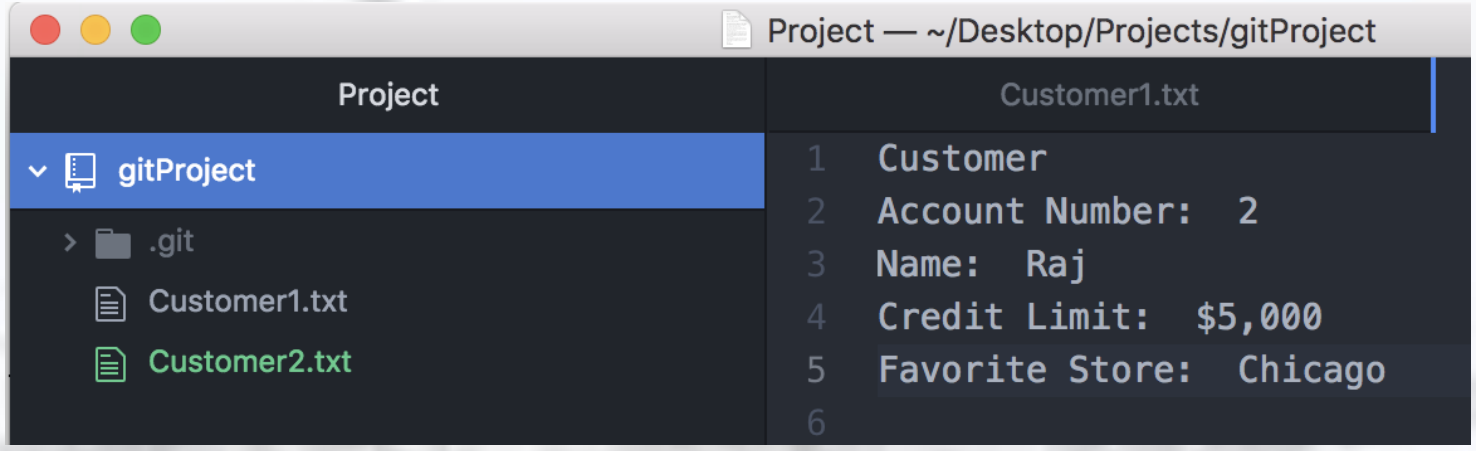
What was changed

Author name,
email and date

Local
Repo



Adding another file



- Create a second file in our project in your text editor
  - I named mine “Customer2.txt”
  - Put some information in your new file

Editing the first file

- Changes to Customer1

Original

Customer1.txt

```
Customer
Account Number:  1
Name:  Bill
Credit Limit:  $1,000
Favorite Store:  Indianapolis
```

The name and the credit limit have been changed

Revised

Customer1.txt

```
Customer
Account Number:  1
Name:  Will
Credit Limit:  $2,000
Favorite Store:  Indianapolis
```

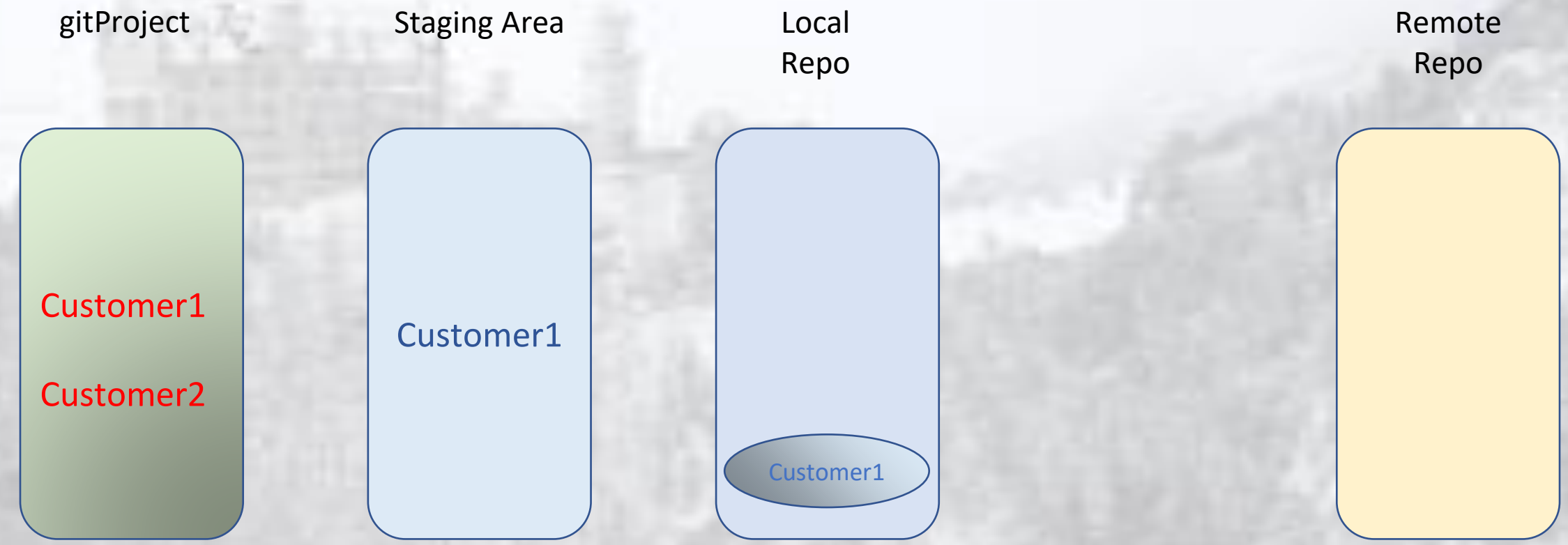
Running $git status

As expected, git tells us that we have one modified file, and one new (untracked) file

gitProject — -bash — 83×24

```
Bills-MacBook-Pro:gitProject supremecommanderstuff$ git status
On branch 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:   Customer1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        Customer2.txt

no changes added to commit (use "git add" and/or "git commit -a")
Bills-MacBook-Pro:gitProject supremecommanderstuff$
```

Current state of the project



What did we change?

What’s different in our modified file?

- In order to see the differences between a file in our working tree and one currently in the staging area, we can use:

$git diff

```
Bills-MacBook-Pro:gitProject supremecommanderstuff$ git diff
diff --git a/Customer1.txt b/Customer1.txt
index 71cdd53..c01ceb7 100644
--- a/Customer1.txt
+++ b/Customer1.txt
@@ -1,5 +1,5 @@
 Customer
 Account Number:  1
-Name:  Bill
-Credit Limit:  $1,000
+Name:  Will
+Credit Limit:  $2,000
 Favorite Store:  Indianapolis
```

Note that $git diff
didn’t mention anything about
Customer2.txt

Why not?

Staging the files

- When we ran $git status (slide 26) it showed one changed file and one untracked file
- We would like to stage these files so that we can commit our changes

- Option 1:
  $git add Customer1.txt Customer2.txt
- Option 2:
  $git add .
  (add .) means to add all new and changed files in the working tree to the staging area
- Option 3:
  $git add Cust*
  The * is a wild card. Any file that begins with “Cust” in the working tree will be staged

Quick Check

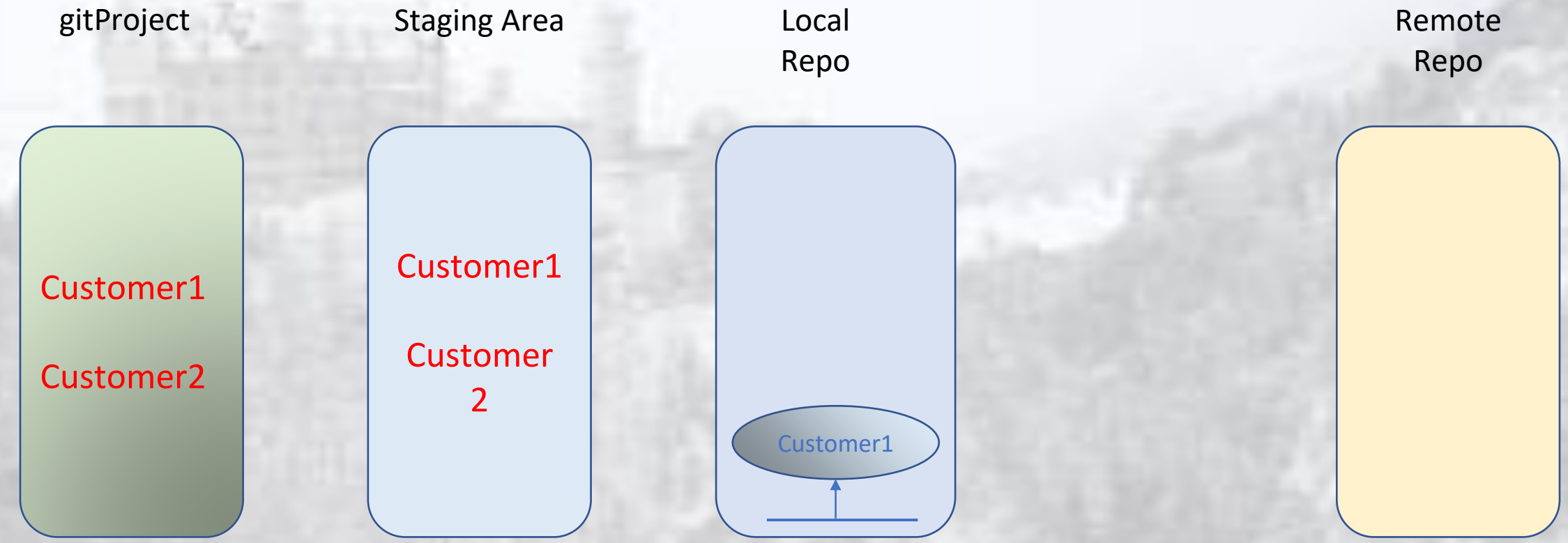
- Stage the 2 files using any of the options on the previous slide
- What will happen when we run $git status

gitProject — -bash — 63×18

```
Bills-MacBook-Pro:gitProject supremecommanderstuff$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   Customer1.txt
        new file:   Customer2.txt

Bills-MacBook-Pro:gitProject supremecommanderstuff$
```

Current state of the project

What would happen if we ran
$git diff



Executing the commit

We commit our files the same way we did before:

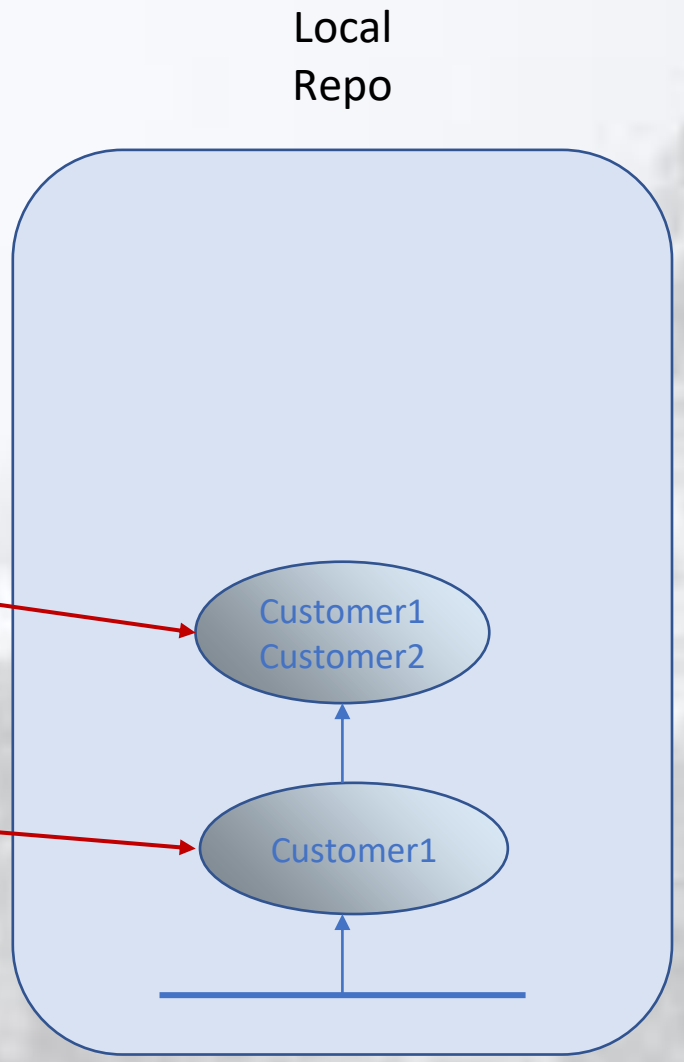
$git commit –m “commit message”

Note that we do not have to tell git which files to commit

It will commit everything in the staging area.

Close-up of the Local Repo

```
Bills-MacBook-Pro:gitProject supremecommanderstuff$ git log
commit 1904155aed0ea9a30f4e219515f8f51bcce8bf9f (HEAD -> master)
Author: Bill Fairfield <bill@fairfieldco.com>
Date:   Sun Jun 10 21:48:21 2018 -0400

    Initial commit for Customer2 and changes to Customer1
    git status

commit f9dde332f9718eb33eac4147daffef083c4cc70b
Author: Bill Fairfield <bill@fairfieldco.com>
Date:   Sat Jun 9 14:10:36 2018 -0400

    add Customer1
Bills-MacBook-Pro:gitProject supremecommanderstuff$
```



Removing a file from the Repo

$git rm filename

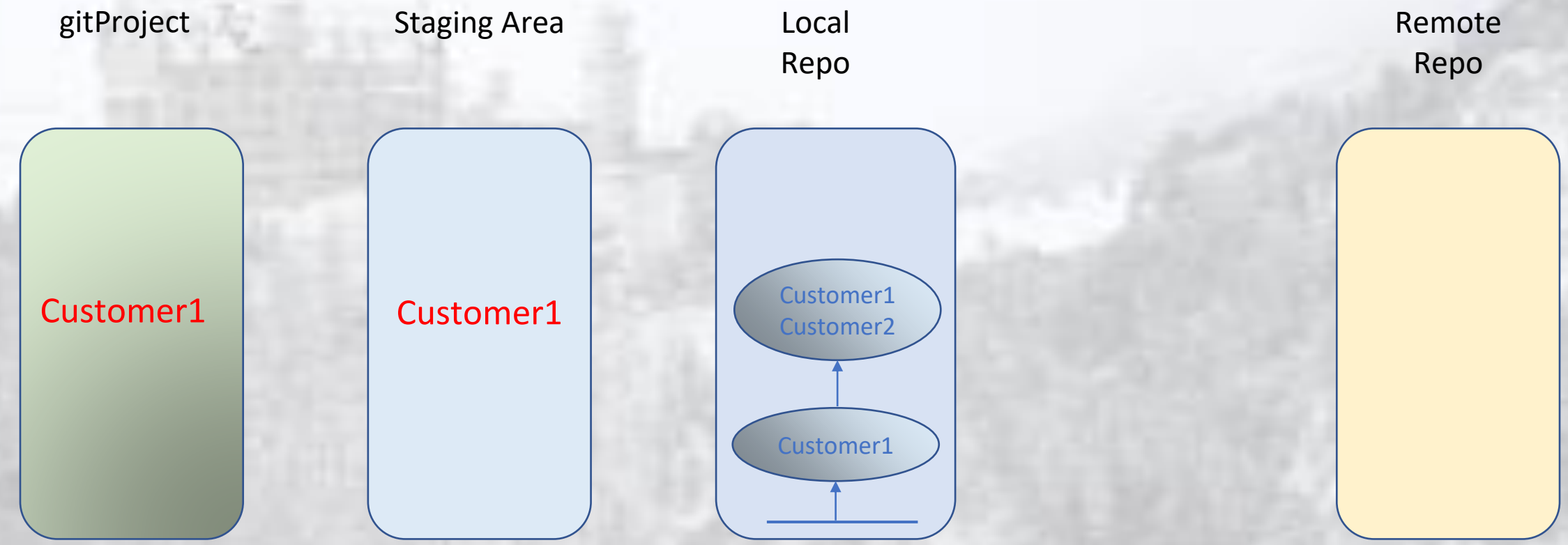
Is the command to remove a file from the repository

This command

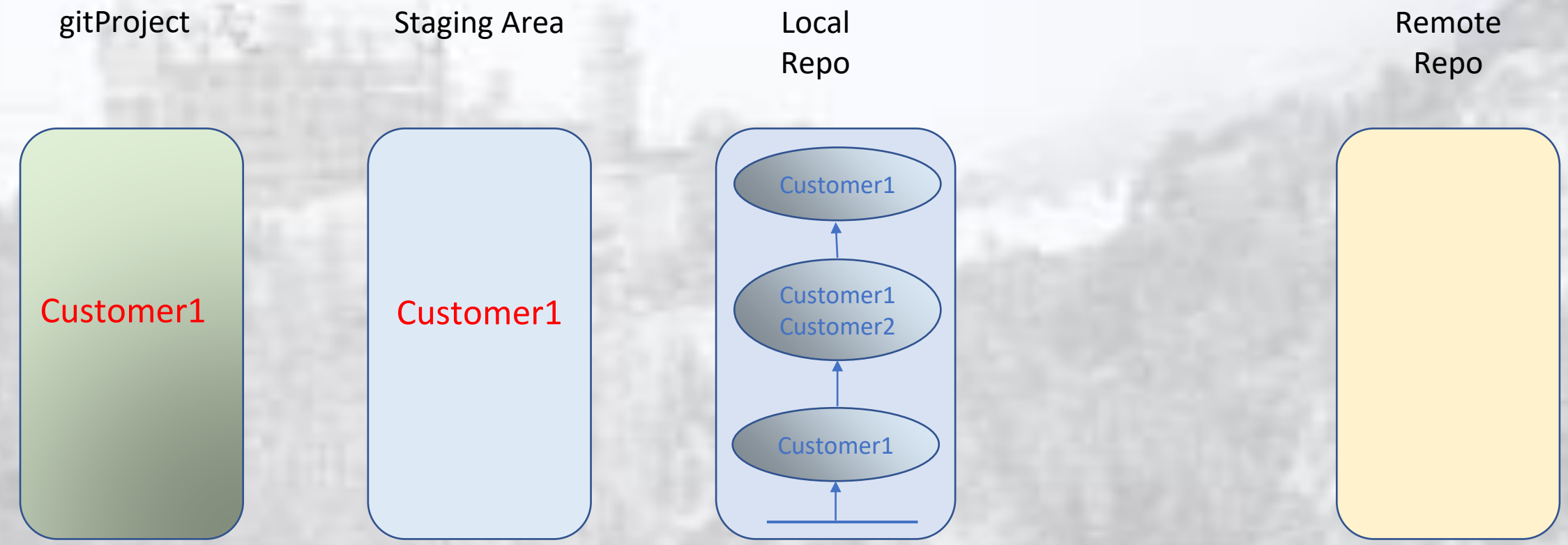
- Removes the file from the Working Tree
- It also stages the removal so that the change removing the file can be committed

- What would the sequence of commands be to remove Customer2 from the repository?

$git rm Customer2.txt



$git commit


gitProject
Staging Area
Local Repo
Remote Repo
Customer1
Customer1
Customer1
Customer1
Customer2
Customer1

Undoing changes in the working tree

- Suppose I have made changes to a file in my working tree and saved them
- Then I decide I don’t like what I just did
- What can I do?

- Option 1

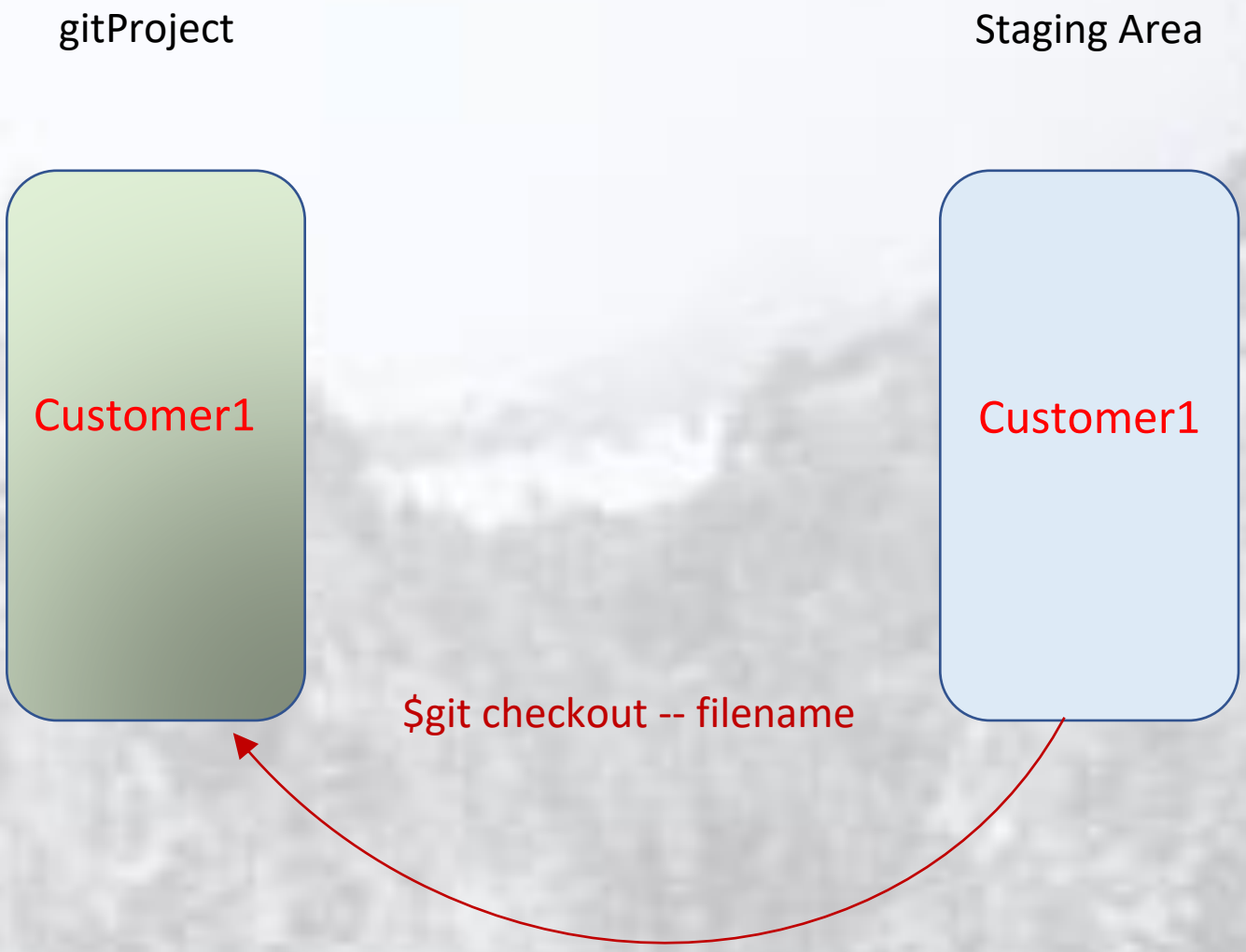
  $git diff

This command will show me the differences between the file in my working tree and the file in the staging area.

But I would have to make the changes manually.

$git checkout --filename

Checkout resets the contents of a file to match the currently staged version of that file



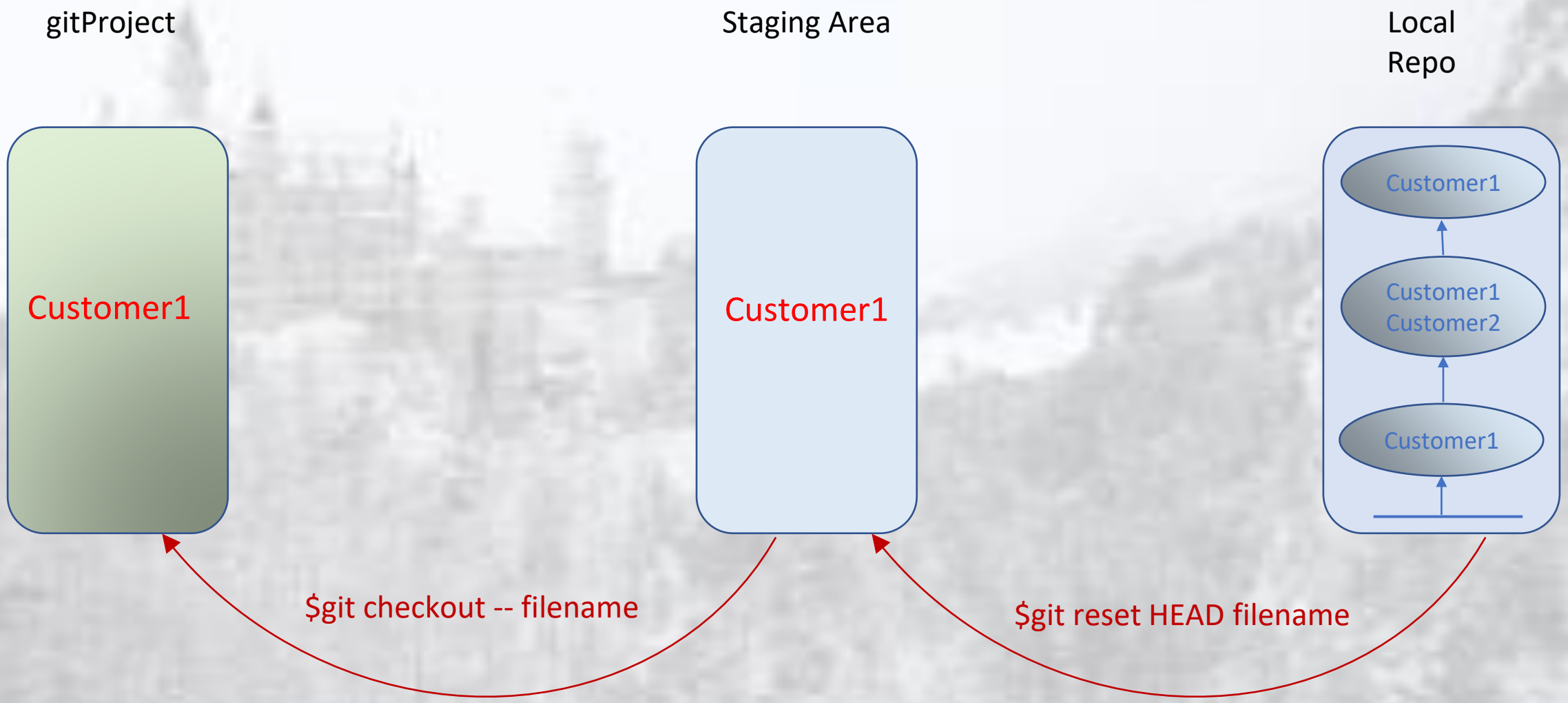
Undoing changes in the staging area

- Suppose I have made changes to a file in my working tree and saved them
- And then I staged them with $git add
- Now I realize that those changes weren't a good idea
- What can I do?

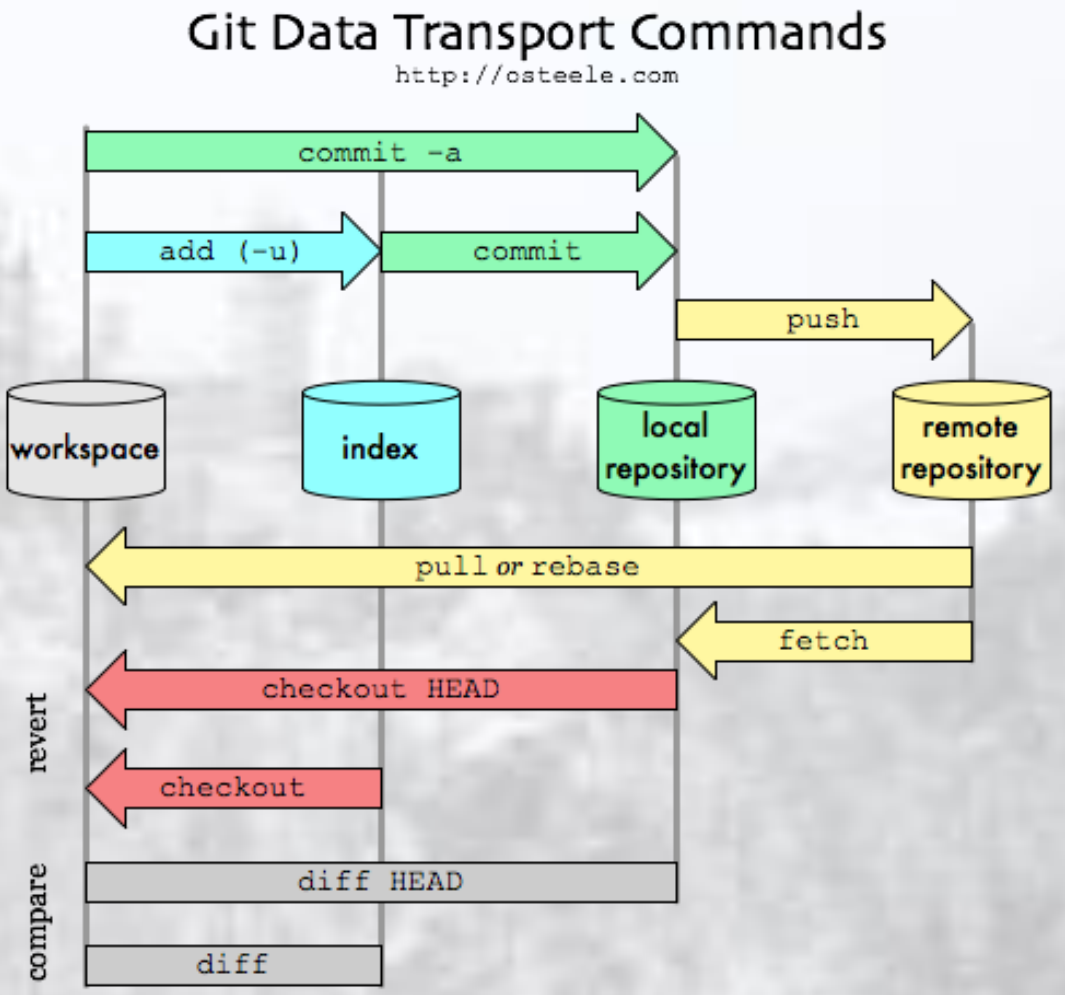
- The command
  $git reset HEAD filename

  Will reset the contents of a file in the staging area to the contents of that file in the local repository

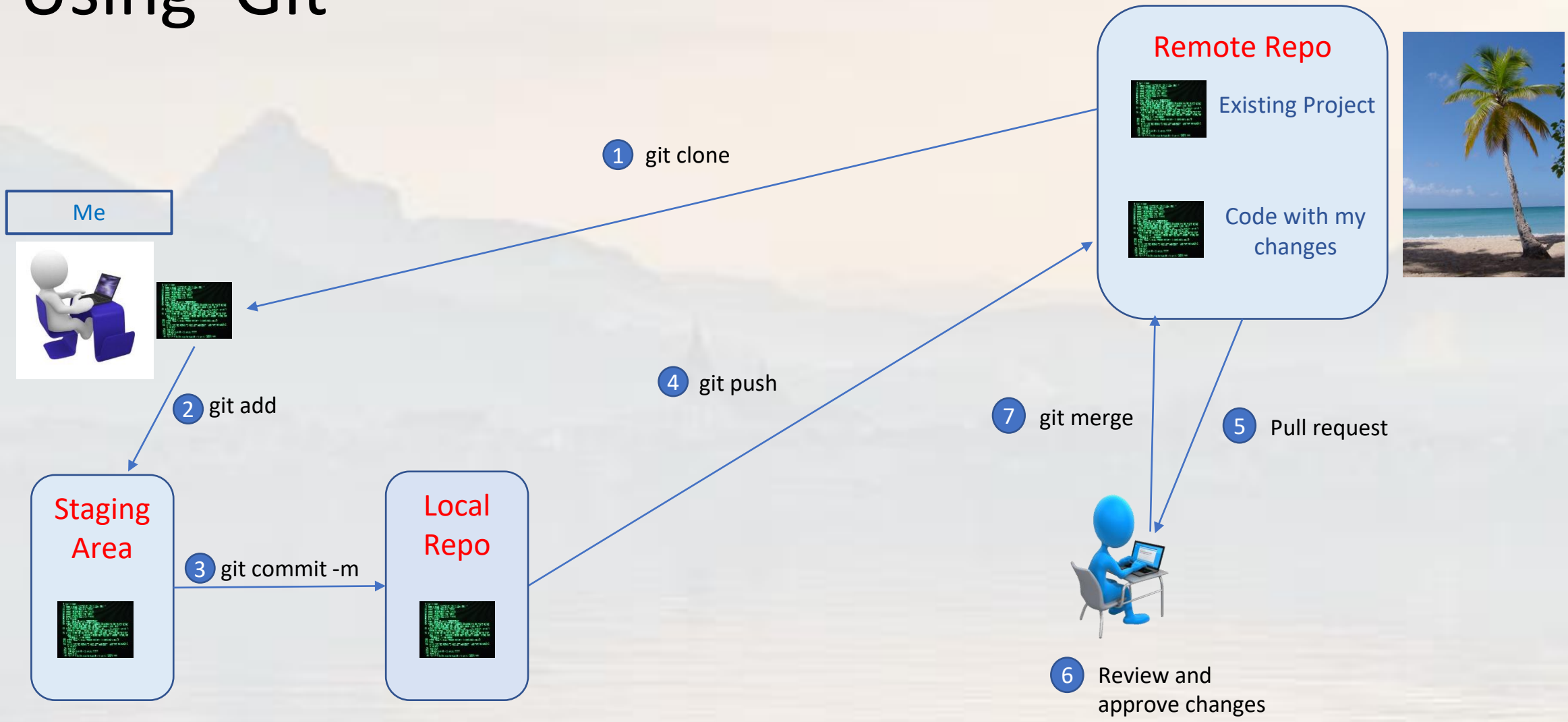
$git checkout --filename



Summary


Git Data Transport Commands
http://osteele.com
commit -a
add (-u)
commit
push
workspace
index
local repository
remote repository
pull or rebase
fetch
checkout HEAD
checkout
revert
diff HEAD
diff
compare

Using Git


Remote Repo
Existing Project
Code with my changes
1 git clone
Me
2 git add
4 git push
7 git merge
5 Pull request
Staging Area
3 git commit -m
Local Repo
6 Review and approve changes