

Module five

Local Repository

1 - Initializing a Local Repository

2 - Going back in history

3 - Changing history

4 - Saving changes

Up to now we have cloned remote repositories created on GitHub to kick start a project. It is also possible to create a brand new Git repository locally on your machine using the git init command.

Initializing a Local Repository

```
15:13:17  learn_git
→ mkdir alphabets
15:18:07  learn_git  233ms
→ cd alphabets
15:18:26  alphabets  124ms
→ git init
Initialized empty Git repository in C:/Users/clldu/OneDrive/learn_git/alphabets/.git/
15:18:40  alphabets  main  242ms
→ ls -a
.  ..  .git
15:23:44  alphabets  main  164ms
→ git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Your new local Git repository is ready to use. You can now start adding files to it.

```
15:29:20  alphabets \main 193ms
→ touch pilots.txt
15:29:34  alphabets \main ?1 363ms
→ git add pilots.txt

15:57:36  alphabets \main +1 173ms
→ git commit -m "Initial commit"
[main (root-commit) 26b09bb] Initial commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 pilots.txt
16:13:53  alphabets \main 15.766s
→ git log --oneline --decorate
26b09bb (HEAD -> main) Initial commit
```

Let's now build some commit history. Make a change to `pilots.txt` by inserting the word "Alpha" on the first line.

```
15:20:04 alphabets \main 173ms
→ touch pilots.txt
15:20:40 alphabets \main ?1 189ms
→ git add pilots.txt
15:20:57 alphabets \main +1 177ms
→ git commit -m "Initial commit"
[main (root-commit) b14f91e] Initial commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 pilots.txt
15:21:21 alphabets \main 221ms
→ git log --oneline --decorate
b14f91e (HEAD -> main) Initial commit
15:21:54 alphabets \main 205ms
→ echo "Alpha" >> pilots.txt
15:30:53 alphabets \main ~1 119ms
→ cat pilots.txt
Alpha
15:31:09 alphabets \main ~1 192ms
→ git add pilots.txt
15:31:55 alphabets \main ~1 161ms
→ git commit -m "Alpha added"
[main 8a7223f] Alpha added
1 file changed, 1 insertion(+)
15:32:38 alphabets \main 167ms
→ git log --oneline --decorate
8a7223f (HEAD -> main) Alpha added
b14f91e Initial commit
```

Bravo and Charlie added to pilots

```
15:47:00  alphabets  ↵main  130ms
→ cat pilots.txt
Alpha
Bravo
Charlie
15:47:07  alphabets  ↵main  138ms
→ git log --oneline --decorate
085f1f9 (HEAD -> main) Charlie added
442308b Bravo added
8a7223f Alpha added
b14f91e Initial commit
15:47:22  alphabets  ↵main  251ms
→ git status
On branch main
nothing to commit, working tree clean
```

A remote repository is not required to follow the exercises in this slides, however for your reference a local Git repository created with `git init` can easily be uploaded to GitHub from the command line. All you have to do is create a new empty repository on GitHub, add it as a new remote and push the contents of the local repo.

After creating a local Git repository with `git init`, you can push it to GitHub by following these steps:

Go to GitHub.

Log in to your GitHub account or create one if you don't have an account.

Click the "+" icon in the top right corner and select "New Repository" to create a new repository on GitHub.

Fill in the repository name, description, and other settings as needed.

Link Local Repository to GitHub Repository:

In your local terminal or command prompt, navigate to the root directory of your local Git repository. Use the following command to set the remote repository URL for your local repository, replacing `username` with your GitHub username and `repository-name` with the name of the GitHub repository you created in step 1:

```
git remote add origin  
https://github.com/username/repository-name.git
```

Commit Your Changes:

Before you can push your code to GitHub, you need to commit your changes. Use the following commands to stage and commit your changes:

```
git add .
```

```
git commit -m "Initial commit" # You can use a  
different commit message
```

Push to GitHub:

Finally, push your local repository to GitHub using the following command:

`git push -u origin main`

This command pushes your code to the main branch (or whatever branch you are currently on) of your GitHub repository. The `-u` flag sets up a tracking relationship between your local branch and the remote branch on GitHub, so you can simply use `git push` in the future to push changes to the same branch.

Authenticate with GitHub:

If you haven't already authenticated with your GitHub account on your local system, Git will prompt you to enter your GitHub username and password or personal access token.

After completing these steps, your local repository will be pushed to your GitHub repository, and you can access it on GitHub's website. Make sure to keep your personal access tokens or credentials secure and use HTTPS or SSH for secure connections, depending on your GitHub settings and preferences.

Create a repository in GitHub

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH git@github.com:linduarte/alpha.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# alpha" >> README.md  
git init  
git add README.md  
git commit -m "first commit"  
git branch -M main  
git remote add origin git@github.com:linduarte/alpha.git  
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin git@github.com:linduarte/alpha.git  
git branch -M main  
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Push to origin

```
 13:55:57  alphabets  main  172ms
→ # Add a new remote identified as origin
 13:56:27  alphabets  main
→ git remote add origin git@github.com:linduarte/alpha.git
 13:58:28  alphabets  main  158ms
→ # Verify the new remote
 13:59:03  alphabets  main
→ git remote --verbose
origin  git@github.com:linduarte/alpha.git (fetch)
origin  git@github.com:linduarte/alpha.git (push)
 13:59:18  alphabets  main  161ms
→ # push the contents of the main branch
 14:00:08  alphabets  main
→ git push origin main
Enter passphrase for key '/c/users/clldu/onedrive/documentos/ssh_keys/.ssh/id_ed25519':
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 16 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (12/12), 880 bytes | 880.00 KiB/s, done.
Total 12 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:linduarte/alpha.git
 * [new branch]      main -> main
```

Going back in history

The `git checkout` command to switch branches and to undo unstaged changes in a file. It can also be used to go back in history in the form `git checkout (commit)` where `(commit)` is the hash (or identifier) of a commit in the revision history.

Detached HEAD

This use of the checkout command will cause the files in the working directory to go back to the state they were when the specified commit took place. Once in this state, called detached HEAD state, any commit as you perform another checkout operation. To preserve any changes you make in a detached HEAD state you need to create a branch.

```
 14:20:02 alphabets main 155ms
→ git log --oneline --decorate
fdf6b7c (HEAD -> main, origin/main) Merge branch 'main' of github.com:linduarte/alpha
eab105e Charlie added
b1398a7 Charlies added
a289b85 Bravo added
472e8ae Alpha added
77473ed Initial commit
 14:21:03 alphabets main 248ms
→ git checkout a289b85
Note: switching to 'a289b85'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -c with the switch command. Example:

```
git switch -c <new-branch-name>
```

Or undo this operation with:

```
git switch -
```

Turn off this advice by setting config variable `advice.detachedHead` to false

```
HEAD is now at a289b85 Bravo added
```

The git checkout output warns of the detached HEAD state. Any new commits made whilst in this state will be lost as soon as you perform another checkout operation. We will demonstrate this with an experiment.

Let's add a new word to `pilots.txt` and commit:

```
 14:37:50  alphabets  detached at a289b85  145ms
→ echo "Delta" >> pilots.txt
 14:38:19  alphabets  detached at a289b85 ~1  138ms
→ git add pilots.txt
 14:38:30  alphabets  detached at a289b85 ~1  171ms
→ git commit -m "Delta added"
[detached HEAD e4bbf9c] Delta added
 1 file changed, 1 insertion(+)
 14:38:49  alphabets  detached at e4bbf9c  218ms
→ git log --oneline --decorate
e4bbf9c (HEAD) Delta added
a289b85 Bravo added
472e8ae Alpha added
77473ed Initial commit
 14:39:07  alphabets  detached at e4bbf9c  252ms
→ cat pilots.txt
Alpha
Bravo
Delta
```

HEAD usual place

```
14:44:13 alphabets detached at e4bbf9c 161ms
→ # Let's move the HEAD back to usual place
14:45:08 alphabets detached at e4bbf9c
→ git checkout main
Warning: you are leaving 1 commit behind, not connected to
any of your branches:

e4bbf9c Delta added

If you want to keep it by creating a new branch, this may be a good time
to do so with:
```

```
git branch <new-branch-name> e4bbf9c
```

```
Switched to branch 'main'
```

```
14:45:22 alphabets main 181ms
→ cat pilots.txt
Alpha
Bravo
Charlie
14:45:50 alphabets main 160ms
→ git log --oneline --decorate
fdf6b7c (HEAD -> main, origin/main) Merge branch 'main' of github.com:linduarte/alpha
eab105e Charlie added
b1398a7 Charlies added
a289b85 Bravo added
472e8ae Alpha added
77473ed Initial commit
```

In order to keep the changes you make whilst in a detached HEAD state you need to create a branch.

```
14:59:00 alphabets ↵detached at a289b85 163ms
→ # To keep the changes in a detached HEAD
14:59:22 alphabets ↵detached at a289b85
→ git checkout a289b85
HEAD is now at a289b85 Bravo added
14:59:39 alphabets ↵detached at a289b85 193ms
→ git checkout -b delta
Switched to a new branch 'delta'
15:00:21 alphabets ↵delta 206ms
→ git branch --all
* delta
  main
  remotes/origin/main
```

Note the `git checkout -b delta` command. The `-b` option creates a new branch (named `delta`) and immediately switches to it. The current branch is now `delta` as you can see from the output of the `git branch --all` command.

Let's make the same change we did earlier to add the word "Delta" to `pilots.txt` and commit:

```
15:27:17 alphabets delta ~1 | ~1 157ms
echo "Delta" >> pilots.txt
15:27:45 alphabets delta ~1 | ~1 116ms
cat pilots.txt
Alpha
Bravo
Delta
15:27:53 alphabets delta ~1 | ~1 172ms
git add pilots.txt
15:28:11 alphabets delta ~1 180ms
git commit -m "Delta added"
[delta 1efae36] Delta added
1 file changed, 1 insertion(+)
15:28:32 alphabets delta 203ms
git log --oneline --decorate
1efae36 (HEAD -> delta) Delta added
a289b85 Bravo added
472e8ae Alpha added
77473ed Initial commit
15:28:57 alphabets delta 218ms
git checkout main
Switched to branch 'main'
15:29:52 alphabets main 188ms
cat pilots.txt
Alpha
Bravo
Charlie
15:30:01 alphabets main 157ms
git checkout delta
Switched to branch 'delta'
15:30:30 alphabets delta 181ms
cat pilots.txt
Alpha
Bravo
Delta
```

To bring the changes into the main branch you just have to merge. In this case Git will flag a conflict in the third line of `pilots.txt` as the content differs between the two branches (we have "Charlie" in main and "Delta" in delta).

Let's switch to main and merge:

```
 15:40:02  █ alphabets  ⏺ delta  █ 186ms
→ git checkout main
Switched to branch 'main'
 15:40:12  █ alphabets  ⏺ main  █ 160ms
→ git merge delta
Auto-merging pilots.txt
CONFLICT (content): Merge conflict in pilots.txt
Automatic merge failed; fix conflicts and then commit the result.
 15:40:29  █ alphabets  ⏺ delta into main ✘ x1 | ✎ x1  █ 211ms
→ cat pilots.txt
Alpha
Bravo
<<<<< HEAD
Charlie
=====
Delta
>>>>> delta
```

We need to resolve the conflict manually. Open `pilots.txt` with VSC and delete the markings added by Git leaving both words ("Charlie" and "Delta") and then commit:

```
15:45:40  alphabets  delta into main x1 | x1 194ms
→ code pilots.txt
15:45:54  alphabets  delta into main x1 | x1 1.796s
→ git add pilots.txt
15:46:55  alphabets  delta into main ~1 190ms
→ git commit -m "Delta added"
[main 92f61c9] Delta added
15:47:17  alphabets  main 220ms
→ git log --oneline --decorate
92f61c9 (HEAD -> main) Delta added
1efae36 (delta) Delta added
fdf6b7c (origin/main) Merge branch 'main' of github.com:linduarte/alpha
eab105e Charlie added
b1398a7 Charlies added
a289b85 Bravo added
472e8ae Alpha added
77473ed Initial commit
15:47:45  alphabets  main 219ms
→ git status
On branch main
nothing to commit, working tree clean
```

Let's understanding what happened

The commit `1efae36` (delta) Delta added is the merge from the delta branch.

The commit `92f61c9` (HEAD -> main) Delta added is the commit following conflict resolution.

This experiment has demonstrated the power of the git checkout command. It allows you to go back to any point in time in the commit history of your source code, make experimental changes in a separate branch and, if required, merge the results back to the main code base.

You can use the git branch -D command to remove a development branch you no longer need:

```
15:58:40  alphabets  main  145ms
→ git branch -D delta
Deleted branch delta (was 1efae36).
15:58:58  alphabets  main  158ms
→ git branch --all
* main
  remotes/origin/main
```

Changing history

There are two Git commands capable of re-writing the commit history of a branch: `git reset` and `git rebase`.

Reset

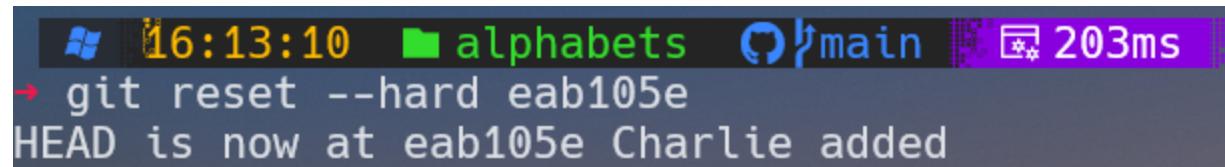
We will demonstrate what `git reset` does with an experiment. Open Git Bash and change to the local "alphabets" repository created earlier.

Dump the log history of the main branch:

```
16:04:52 alphabets main 176ms
→ git log --oneline --decorate
92f61c9 (HEAD -> main) Delta added
1efae36 Delta added
fdf6b7c (origin/main) Merge branch 'main' of github.com:linduarte/alpha
eab105e Charlie added
b1398a7 Charlies added
a289b85 Bravo added
472e8ae Alpha added
77473ed Initial commit
16:05:08 alphabets main 217ms
→ cat pilots.txt
Alpha
Bravo
Charlie
Delta
```

Suppose you want to reset the history of commits to the point where you added the word "Charlie" (hash eab105e in my history log, it will be something else in your repo).

All you have to do is typing the following command replacing the hash value with the one in your history log:



```
16:13:10 alphabets main 203ms
→ git reset --hard eab105e
HEAD is now at eab105e Charlie added
```

A screenshot of a Windows terminal window. The title bar shows the time as 16:13:10, the directory as alphabets, and the branch as main. The system tray indicates a battery level of 203ms. The command entered is 'git reset --hard eab105e'. The output shows that HEAD is now at the commit 'eab105e Charlie added'.

Now check again the history log and you will find that the last two commits have been removed. The `pilots.txt` file contents have gone back to the point prior to the merging operation we performed in the previous slide:

```
 16:17:42  alphabets  main  218ms
→ git log --oneline --decorate
eab105e (HEAD -> main) Charlie added
a289b85 Bravo added
472e8ae Alpha added
77473ed Initial commit
 16:17:54  alphabets  main  229ms
→ cat pilots.txt
Alpha
Bravo
Charlie
```

Attention

Although removing unwanted commits can be useful in some situations, it must be done with extreme caution. Warning: *Never change the commit history of shared branches* when collaborating with other users as it will cause them a lot of problems when you push the changes back to a shared public repository. Only use this form of reset if you want to remove unwanted commits from a local private branch.

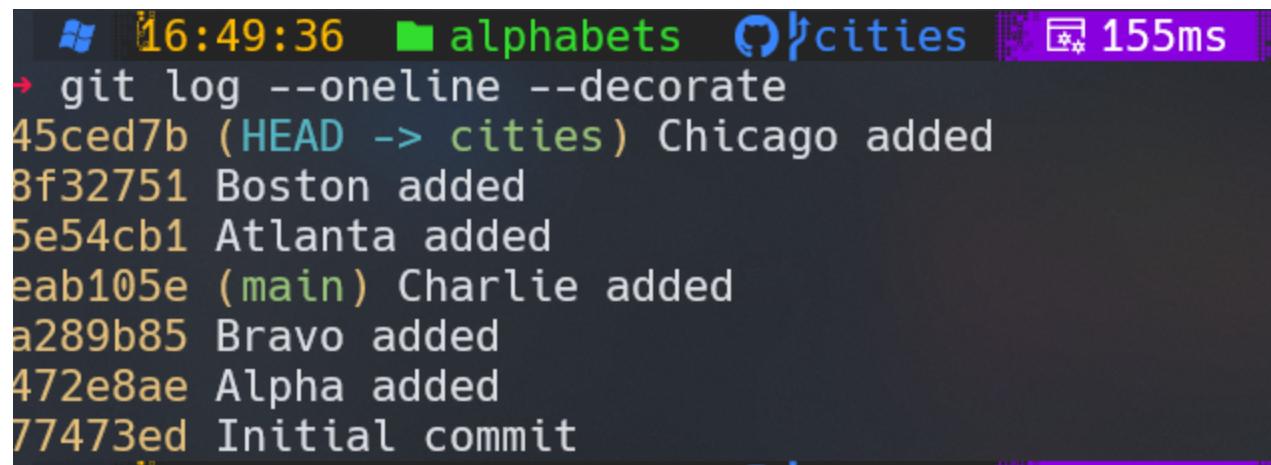
Rebase

The command `git rebase` can be used in place of `git merge` to integrate the work you have done in separate branches. Whereas the merge operation preserves history, the `rebase operation can modify the target branch history` by inserting intermediary commits.

Let's explore the difference between git merge and git rebase.

```
 16:40:08 └─ alphabets ┌─ main ┌─ 158ms ┌─
→ git checkout -b cities
Switched to a new branch 'cities'
 16:40:28 └─ alphabets ┌─ cities ┌─ 193ms ┌─
→ touch cities.txt
 16:40:44 └─ alphabets ┌─ cities ┌─ ?1 ┌─ 173ms ┌─
→ echo "Atlanta" >> cities.txt
 16:41:22 └─ alphabets ┌─ cities ┌─ ?1 ┌─ 94ms ┌─
→ git add cities.txt
 16:41:35 └─ alphabets ┌─ cities ┌─ +1 ┌─ 182ms ┌─
→ git commit -m "Atlanta added"
[cities 5e54cb1] Atlanta added
1 file changed, 1 insertion(+)
create mode 100644 cities.txt
 16:41:56 └─ alphabets ┌─ cities ┌─ 219ms ┌─
→ echo "Boston" >> cities.txt
 16:42:44 └─ alphabets ┌─ cities ┌─ ~1 ┌─ 117ms ┌─
→ git add cities.txt
 16:42:55 └─ alphabets ┌─ cities ┌─ ~1 ┌─ 201ms ┌─
→ git commit -m "Boston added"
[cities 8f32751] Boston added
1 file changed, 1 insertion(+)
 16:43:25 └─ alphabets ┌─ cities ┌─ 191ms ┌─
→ echo "Chicago" >> cities.txt
 16:43:46 └─ alphabets ┌─ cities ┌─ ~1 ┌─ 132ms ┌─
→ git add cities.txt
 16:44:00 └─ alphabets ┌─ cities ┌─ ~1 ┌─ 206ms ┌─
→ git commit -m "Chicago added"
[cities 45ced7b] Chicago added
1 file changed, 1 insertion(+)
```

After the previous process we get:



A screenshot of a Windows command-line interface (cmd) window. The title bar shows the path 'alphabets \cities' and the status bar indicates it's running at 155ms. The main content is a git log command output:

```
16:49:36 alphabets \cities 155ms
→ git log --oneline --decorate
45ced7b (HEAD -> cities) Chicago added
8f32751 Boston added
5e54cb1 Atlanta added
eab105e (main) Charlie added
a289b85 Bravo added
472e8ae Alpha added
77473ed Initial commit
```

Now we added Delta, Echo and Foxtrot to main branch

```
 16:54:30 alphabets Q\cities 164ms
→ git checkout main
Switched to branch 'main'
 16:54:44 alphabets Q\main 173ms
→ git log --oneline --decorate
eab105e (HEAD -> main) Charlie added
a289b85 Bravo added
472e8ae Alpha added
77473ed Initial commit
 16:55:03 alphabets Q\main 204ms
→ echo "Delta" >> pilots.txt
 16:56:15 alphabets Q\main ~1 103ms
→ git add pilots.txt
 16:56:27 alphabets Q\main ~1 211ms
→ git commit -m "Delta added"
[main b2b8dbe] Delta added
1 file changed, 1 insertion(+)
 16:56:46 alphabets Q\main 220ms
→ echo "Echo" >> pilots.txt
 16:57:16 alphabets Q\main ~1 117ms
→ git add pilots.txt
 16:57:29 alphabets Q\main ~1 189ms
→ git commit -m "Echo added"
[main 9d08f2e] Echo added
1 file changed, 1 insertion(+)
 16:57:46 alphabets Q\main 207ms
→ echo "Foxtrot" >> pilots.txt
 16:58:09 alphabets Q\main ~1 92ms
→ git add pilots.txt
 16:58:21 alphabets Q\main ~1 175ms
→ git commit -m "Foxtrot added"
[main 87d5458] Foxtrot added
1 file changed, 1 insertion(+) 
```

Suppose we want to integrate the work we have done on the cities branch on to the main branch. This time, instead of **git merge**, we will use **git rebase**. Let's see what happens:

```
17:02:58 alphabets ①\main 175ms
→ git log --oneline --decorate
87d5458 (HEAD -> main) Foxtrot added
9d08f2e Echo added
b2b8dbe Delta added
eab105e Charlie added
a289b85 Bravo added
472e8ae Alpha added
77473ed Initial commit
17:03:14 alphabets ①\main 221ms
→ git rebase cities
Successfully rebased and updated refs/heads/main.
17:06:06 alphabets ①\main 278ms
→ git log --oneline --decorate
dcb98d3 (HEAD -> main) Foxtrot added
c30ff4c Echo added
4ac765a Delta added
45ced7b (cities) Chicago added
8f32751 Boston added
5e54cb1 Atlanta added
eab105e Charlie added
a289b85 Bravo added
472e8ae Alpha added
77473ed Initial commit
```

With `git merge` the commits would have been added to the history of the main branch at the point of merge, leaving the previous history intact. The rebase operation instead has replayed the commits we did on the cities branch on top of the main branch **effectively modifying its history** as if we had done all the work sequentially on the master branch.

Warning

As we have already stated, changing the commit history of a public shared branch will confuse and potentially cause errors when with other users. Only use rebase instead of merge in short-lived local private branches where you want to see the commit history of the feature branch replayed onto the target branch. Never use rebase on public shared branches that are pushed to remote repositories

Saving changes

The `git stash` command can be used to "stash away" half-baked changes are not prepared to commit yet but want to keep for later.

Let's practice on main branch adding some words and keep them staged:

```
 17:18:00 alphabets main 176ms
→ ls
cities.txt pilots.txt
 17:18:11 alphabets main 156ms
→ git status
On branch main
nothing to commit, working tree clean
 17:18:19 alphabets main 159ms
→ echo "Sundance" >> cities.txt
 17:21:11 alphabets main ~1 95ms
→ echo "Laura" >> pilots.txt
 17:22:05 alphabets main ~2 138ms
→ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   cities.txt
    modified:   pilots.txt

no changes added to commit (use "git add" and/or "git commit -a")
 17:22:15 alphabets main ~2 159ms
→ git add *.txt
 17:26:07 alphabets main ~2 157ms
→ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   cities.txt
    modified:   pilots.txt
```

Suppose now you decide to start some other work from a clean working directory but without loosing the changes you have made so far. You can simply stash the changes away by typing:

```
 17:30:26  alphabets  main ~2  159ms
→ git stash
Saved working directory and index state WIP on main: dcba98d3 Foxtrot added
 17:30:59  alphabets  main 1  267ms
→ git status
On branch main
nothing to commit, working tree clean
 17:31:17  alphabets  main 1  138ms
→ git stash list
stash@{0}: WIP on main: dcba98d3 Foxtrot added
 17:31:50  alphabets  main 1  232ms
→ git stash show
cities.txt | 1 +
pilots.txt | 1 +
2 files changed, 2 insertions(+)
```

The working directory and index are now clean;

But your changes have not been lost. You can view any modifications stashed away with the following commands:`git stash list` and `git stash show`.

When you decide to restore the work you have stashed away you can do so by entering the command `git stash apply`:

```
17:38:37 alphabets \main 1 188ms
→ git stash apply
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   cities.txt
    modified:   pilots.txt

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