

Module six

The repository database

Every time you initialize a local or remote Git repository a sub-directory named *.git* is automatically created for you. The *.git* directory is where Git stores all the commit history and metadata for a project. In this section we will explore the *.git* directory in some detail.

Start Git Bash and change to the open-website project working directory we created previously. List the contents of the .git directory:

```
15:37:22 learn_git
→ cd alphabets
15:37:31 alphabets main 1 116ms
→ ls .git
COMMIT_EDITMSG FETCH_HEAD HEAD ORIG_HEAD config description hooks index info logs objects packed-refs refs
15:37:51 alphabets main 1 147ms
→ cat .git/HEAD
ref: refs/heads/main
```

The refs directory contains references to commits for local and remote tracking branches.

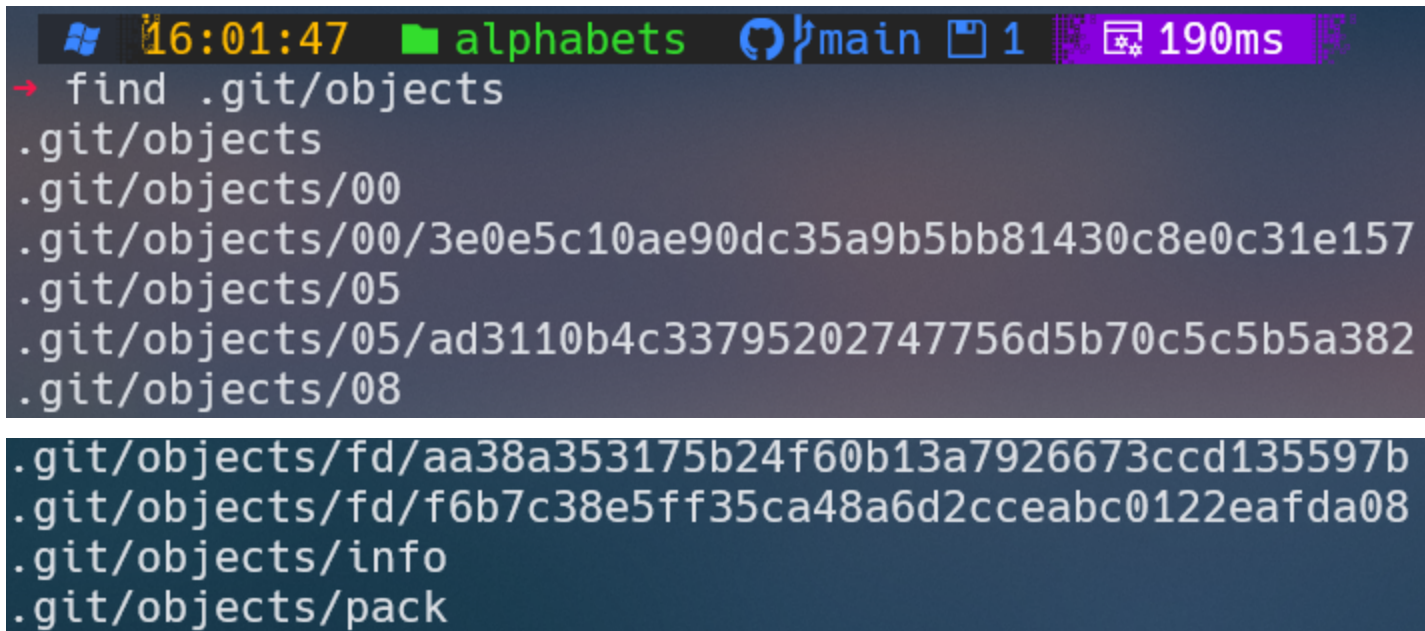
```
15:45:52 alphabets main 1 177ms
→ find .git/refs
.git/refs
.git/refs/heads
.git/refs/heads/cities
.git/refs/heads/main
.git/refs/remotes
.git/refs/remotes/origin
.git/refs/remotes/origin/main
.git/refs/stash
.git/refs/tags
```

To see the reference to the most recent commit in the main branch type:

```
15:50:17 alphabets main 1 159ms
→ cat .git/refs/heads/main
beede61d0306d229c091395f28af95b5bc1ea70b

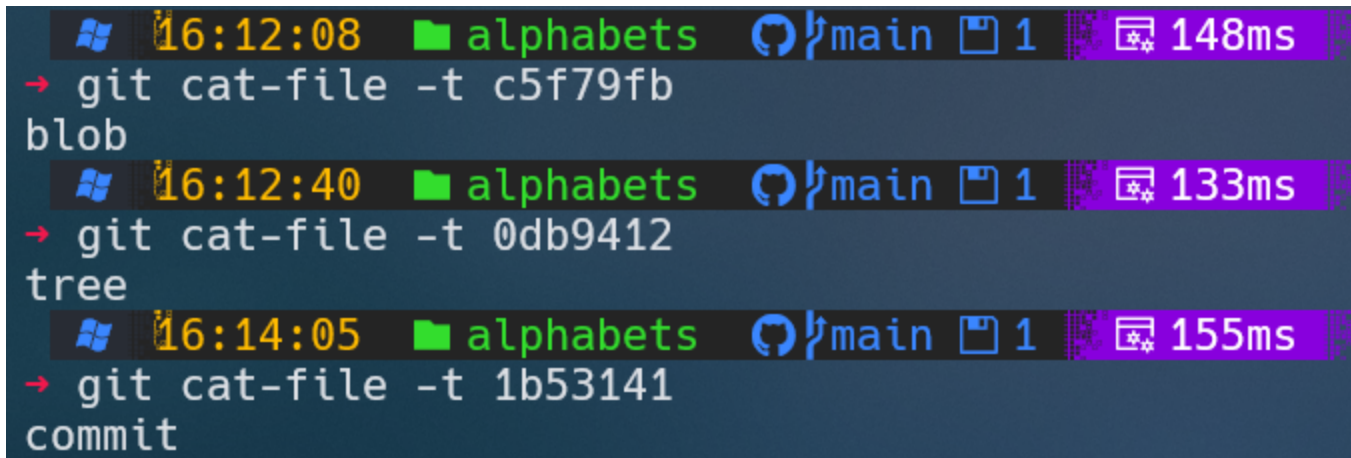
15:50:40 alphabets main 1 142ms
→ git log --oneline --max-count=1
beede61 (HEAD -> main) Add gitignore file to exclude idea/ directory
```

The actual commits are stored in the objects directory. Git stores four types of objects in this directory: commits, trees (directories), blobs (files) and tags.

A screenshot of a Windows command prompt window. The title bar shows the Windows logo, the time 16:01:47, the folder name alphabets, the current branch main, a file icon with the number 1, and a performance indicator of 190ms. The command prompt shows the command 'find .git/objects' being executed. The output lists the contents of the .git/objects directory, including subdirectories and files with their SHA-1 hashes.

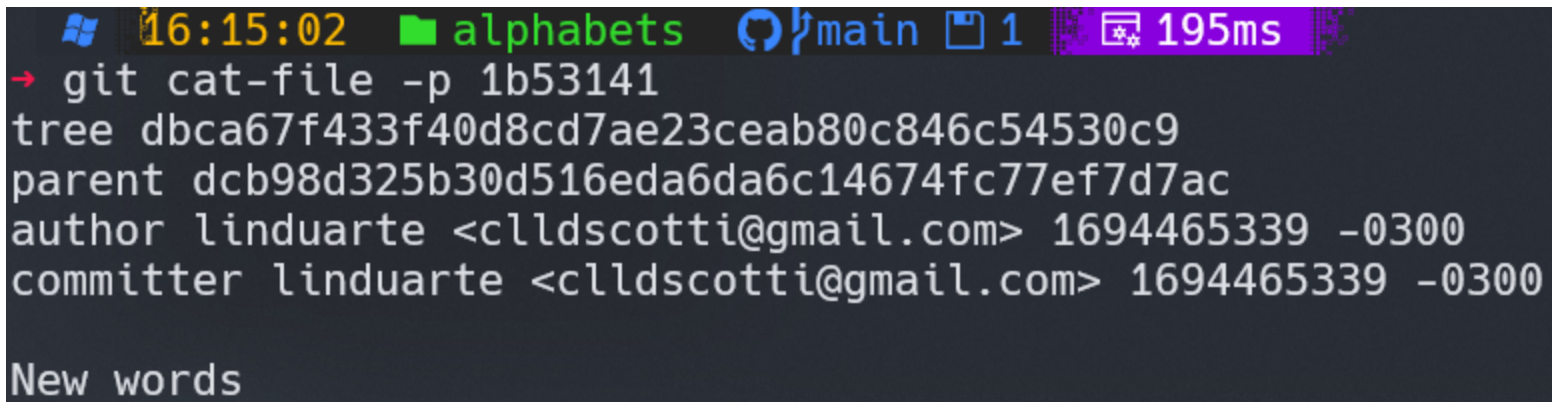
```
→ find .git/objects
.git/objects
.git/objects/00
.git/objects/00/3e0e5c10ae90dc35a9b5bb81430c8e0c31e157
.git/objects/05
.git/objects/05/ad3110b4c33795202747756d5b70c5c5b5a382
.git/objects/08
.git/objects/fd/aa38a353175b24f60b13a7926673ccd135597b
.git/objects/fd/f6b7c38e5ff35ca48a6d2cceabc0122eafda08
.git/objects/info
.git/objects/pack
```

You can find out the type of each of the objects listed above using the `git cat-file` command with the `-t` option. You only need to specify the first seven characters of the object's hash.



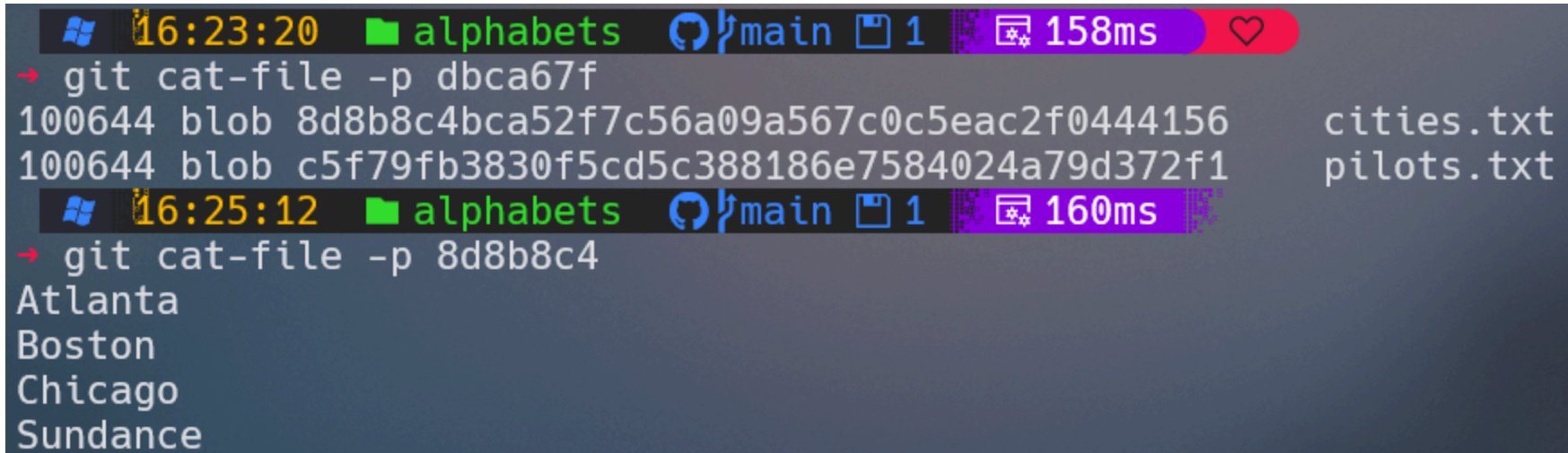
```
16:12:08 alphabets main 1 148ms  
→ git cat-file -t c5f79fb  
blob  
16:12:40 alphabets main 1 133ms  
→ git cat-file -t 0db9412  
tree  
16:14:05 alphabets main 1 155ms  
→ git cat-file -t 1b53141  
commit
```

The git cat-file output confirms that it is a commit object. You can look at its contents using the -p option

A terminal window screenshot with a dark background. The title bar shows a Windows icon, the time 16:15:02, the directory alphabets, the branch main, a file icon with the number 1, and a refresh icon with 195ms. The command prompt shows a red arrow followed by 'git cat-file -p 1b53141'. The output is a commit object in plain text, showing the tree hash, parent hash, author information, and committer information. At the bottom, the text 'New words' is partially visible.

```
16:15:02 alphabets main 1 195ms  
→ git cat-file -p 1b53141  
tree dbca67f433f40d8cd7ae23ceab80c846c54530c9  
parent dcb98d325b30d516eda6da6c14674fc77ef7d7ac  
author linduarte <clldscotti@gmail.com> 1694465339 -0300  
committer linduarte <clldscotti@gmail.com> 1694465339 -0300  
  
New words
```

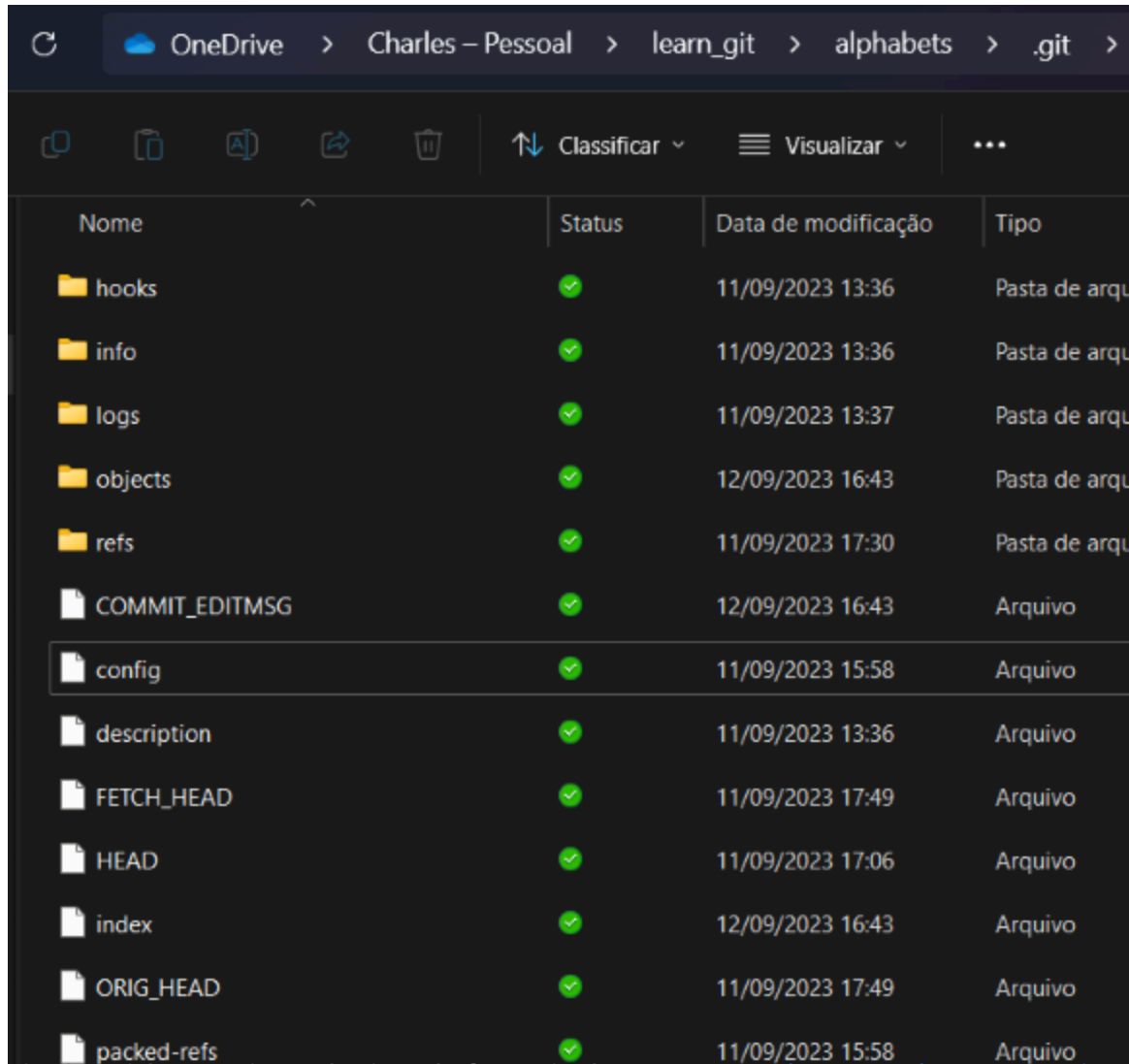

If we use cat-file pointing to the tree we get:

A terminal window with a dark background. The top status bar shows the time 16:23:20, the directory alphabets, the branch main, a file icon with the number 1, a performance indicator of 158ms, and a heart icon. The command 'git cat-file -p dbca67f' is entered. The output shows two entries: '100644 blob 8d8b8c4bca52f7c56a09a567c0c5eac2f0444156 cities.txt' and '100644 blob c5f79fb3830f5cd5c388186e7584024a79d372f1 pilots.txt'. Below this, the time is 16:25:12, and the command 'git cat-file -p 8d8b8c4' is entered. The output lists the cities: Atlanta, Boston, Chicago, and Sundance.

```
16:23:20 alphabets main 1 158ms
→ git cat-file -p dbca67f
100644 blob 8d8b8c4bca52f7c56a09a567c0c5eac2f0444156 cities.txt
100644 blob c5f79fb3830f5cd5c388186e7584024a79d372f1 pilots.txt
16:25:12 alphabets main 1 160ms
→ git cat-file -p 8d8b8c4
Atlanta
Boston
Chicago
Sundance
```

We have followed the trail from HEAD all the way to the last commit object on main and the associated directory and files. This is how Git stores the project history and is able to retrieve any previous versions.

Let's take a look of the "alphabets" site showing folders and files.



Nome	Status	Data de modificação	Tipo
hooks	✓	11/09/2023 13:36	Pasta de arqu
info	✓	11/09/2023 13:36	Pasta de arqu
logs	✓	11/09/2023 13:37	Pasta de arqu
objects	✓	12/09/2023 16:43	Pasta de arqu
refs	✓	11/09/2023 17:30	Pasta de arqu
COMMIT_EDITMSG	✓	12/09/2023 16:43	Arquivo
config	✓	11/09/2023 15:58	Arquivo
description	✓	11/09/2023 13:36	Arquivo
FETCH_HEAD	✓	11/09/2023 17:49	Arquivo
HEAD	✓	11/09/2023 17:06	Arquivo
index	✓	12/09/2023 16:43	Arquivo
ORIG_HEAD	✓	11/09/2023 17:49	Arquivo
packed-refs	✓	11/09/2023 15:58	Arquivo

The config file contains project-specific configuration. The index file is the staging area where changes are grouped before doing a commit. The hooks directory contains scripts that are executed before or after a specific Git command. The info directory contains additional information about the repository. The logs directory contains the history of each branch as displayed by the git log command.

A More Sophisticated History View

We have often used the `git log` command with the `--oneline` and `--decorate` options to view the history of commits in a project. However there is a way to type less and get more information out of `git log` by using aliases. An alias is an alternative name that you can give to a Git command.

Aliases can be setup in the Git configuration file `.gitconfig` located in your home directory. Start Git Bash and enter the following commands to locate it:

```
/c/Users/clldu/OneDrive/learn_git
16:50:27 learn_git 110ms
→ cd /c/users/clldu
16:50:48 ~ 115ms
→ ls -a .gitconfig
.gitconfig
16:51:17 ~ 140ms
→ code .gitconfig
```

Add an [alias] entry named hist:

[alias]

```
hist=log --pretty=format:'%C(yellow)%h %Cred%ad %Cblue%an%Cgreen%d
%Creset%s' --date=short
```

To try the new alias change to a Git project working directory and type git hist:

```
17:21:30 alphabets main 1 171ms
→ git hist
beede61 2023-09-12 linduarte (HEAD -> main) Add gitignore file to exclude idea/ directory
47a81de 2023-09-11 linduarte (origin/main) Merge branch 'main' of github.com:linduarte/alpha
1b53141 2023-09-11 linduarte New words
dcb98d3 2023-09-11 linduarte Foxtrot added
c30ff4c 2023-09-11 linduarte Echo added
4ac765a 2023-09-11 linduarte Delta added
45ced7b 2023-09-11 linduarte (cities) Chicago added
8f32751 2023-09-11 linduarte Boston added
5e54cb1 2023-09-11 linduarte Atlanta added
fdf6b7c 2023-09-11 linduarte Merge branch 'main' of github.com:linduarte/alpha
eab105e 2023-09-11 linduarte Charlie added
b1398a7 2023-09-11 linduarte Charlies added
a289b85 2023-09-11 linduarte Bravo added
472e8ae 2023-09-11 linduarte Alpha added
77473ed 2023-09-11 linduarte Initial commit
```

You can still append additional switches such as the `--max-count` to limit the output to the more recent commits only:

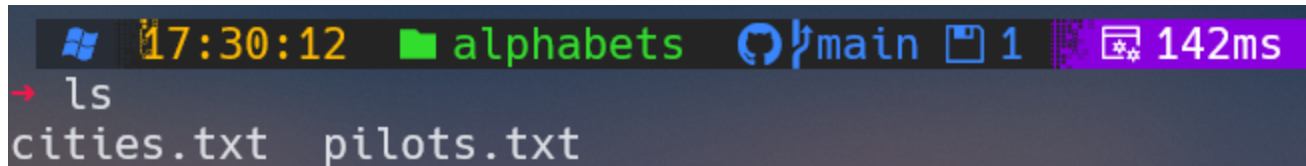
```
17:08:06 alphabets main 1 251ms
→ git hist --max-count=3
beede61 2023-09-12 linduarte (HEAD -> main) Add gitignore file to exclude idea/ directory
47a81de 2023-09-11 linduarte (origin/main) Merge branch 'main' of github.com:linduarte/alpha
1b53141 2023-09-11 linduarte New words
```

Ignoring Files

In a project there are often files that you do not want Git to track because there is no interest in keeping a version history of them. They can be binary files produced by a compiler or other temporary files generated automatically by code editors and other tools. You do not want all this "noise" in your working directory to go into your project history repository.

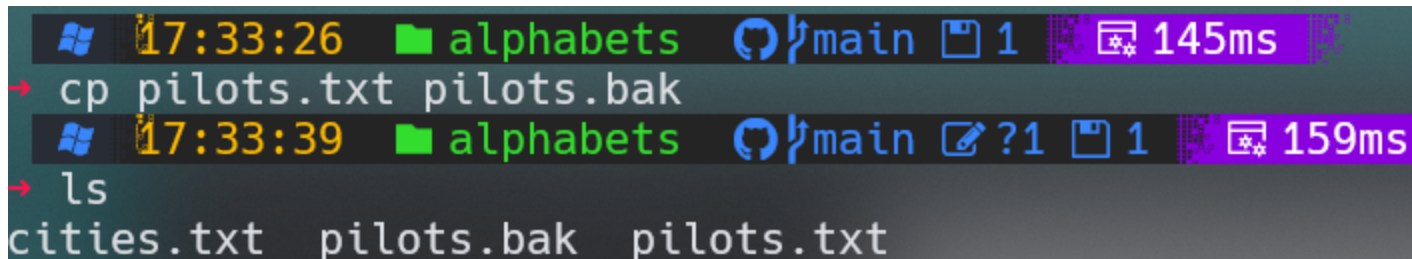
To prevent Git from tracking such files and directories all you have to do is to create a text file named `.gitignore` in the root of the working directory of your project. Inside `.gitignore` you can specify the names of files and directories you want Git to ignore.

Suppose we want to ignore all files with a `.bak` extension. Start Git Bash and change to the alphabets project working directory:



```
17:30:12 alphabets main 1 142ms  
→ ls  
cities.txt pilots.txt
```

Create a file with the `.bak` extension by copying an existing file:



```
17:33:26 alphabets main 1 145ms  
→ cp pilots.txt pilots.bak  
17:33:39 alphabets main ?1 1 159ms  
→ ls  
cities.txt pilots.bak pilots.txt
```

Git will normally track the .bak file we just added:

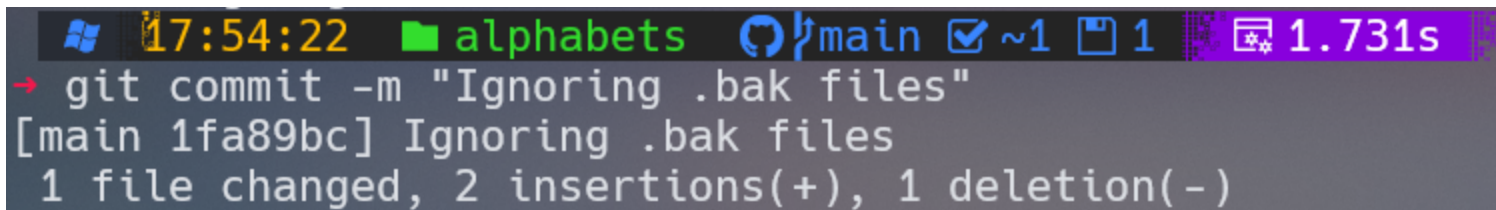
```
17:38:36 alphabets main ?1 1 132ms
→ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        pilots.bak

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

To tell Git to stop tracking files with a .bak extension first create a .gitignore file:

```
17:42:19 alphabets main ?1 1 147ms
→ touch .gitignore
17:42:50 alphabets main ?1 1 158ms
→ ls -a
.  ..  .git  .gitignore  .idea  cities.txt  pilots.bak  pilots.txt
17:42:53 alphabets main ?1 1 159ms
→ code .gitignore
17:46:40 alphabets main ?1 1 1.691s
→ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
```

As you can see git status is now ignoring `*.bak` files. It is good practice to commit the `.gitignore` file to the repository:



```
17:54:22 alphabets main ~1 1 1.731s  
→ git commit -m "Ignoring .bak files"  
[main 1fa89bc] Ignoring .bak files  
1 file changed, 2 insertions(+), 1 deletion(-)
```

You can also instruct Git to ignore an entire directory. Let's create a new directory called temp containing two files using the command sequence:

```
18:02:06 alphabets main ?1 ~1 1 201ms  
→ mkdir temp  
18:02:32 alphabets main ?1 ~1 1 157ms  
→ touch temp/temp.log  
18:02:57 alphabets main ?2 ~1 1 144ms  
→ touch temp/temp.old
```

```
18:06:20 alphabets main ?1 ~1 1 160ms  
→ 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:   .gitignore  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    temp/  
  
no changes added to commit (use "git add" and/or "git commit -a")
```


Open .gitignore in a text editor and add the directory name to the list.

```
18:12:36 alphabets main ~1 1 132ms
→ 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:   .gitignore

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

Commit the changes to .gitignore:

```
18:18:20 alphabets main ~1 1 155ms
→ git add .gitignore
18:18:35 alphabets main ~1 1 157ms
→ git commit -m "Ignoring temp directory"
[main ce4becb] Ignoring temp directory
1 file changed, 2 insertions(+), 2 deletions(-)
18:19:02 alphabets main 1 189ms
→ git status
On branch main
nothing to commit, working tree clean
```

After all those steps let's check the final history:

```
18:21:22 alphabets main 1 137ms  
→ git hist --max-count=3  
ce4becb 2023-09-13 linduarte (HEAD -> main) Ignoring temp directory  
1fa89bc 2023-09-13 linduarte Ignoring .bak files  
beede61 2023-09-12 linduarte Add gitignore file to exclude idea/ directory
```

Git workflows

