Introduction to Git and GitHub

Elham E Khoda
San Diego Supercomputing Center, UCSD

Adopted from previous summer institutes. Thanks to ***Martin Kandes*** *and* ***Mahidhar Tatineni***

UC San Diego

HPC and Data Science Summer Institute 2024
August 7, 2024

SDSC
SAN DIEGO SUPERCOMPUTER CENTER

Go to menti.com

Code: 88 22 23 8

Or

https://www.menti.com/alpajj2epzh3

Today’s Session

- An Overview of Version Control
- About git
- Getting Started with git
- Working with git
- Getting Started with GitHub
- Summary and Conclusion
- Q&A

Version Control

What is version control?

Version Control

What is version control?

*The **practice of tracking and managing changes** to a file or set of files over time so that you can recall specific versions later, if needed*

What do we version control?

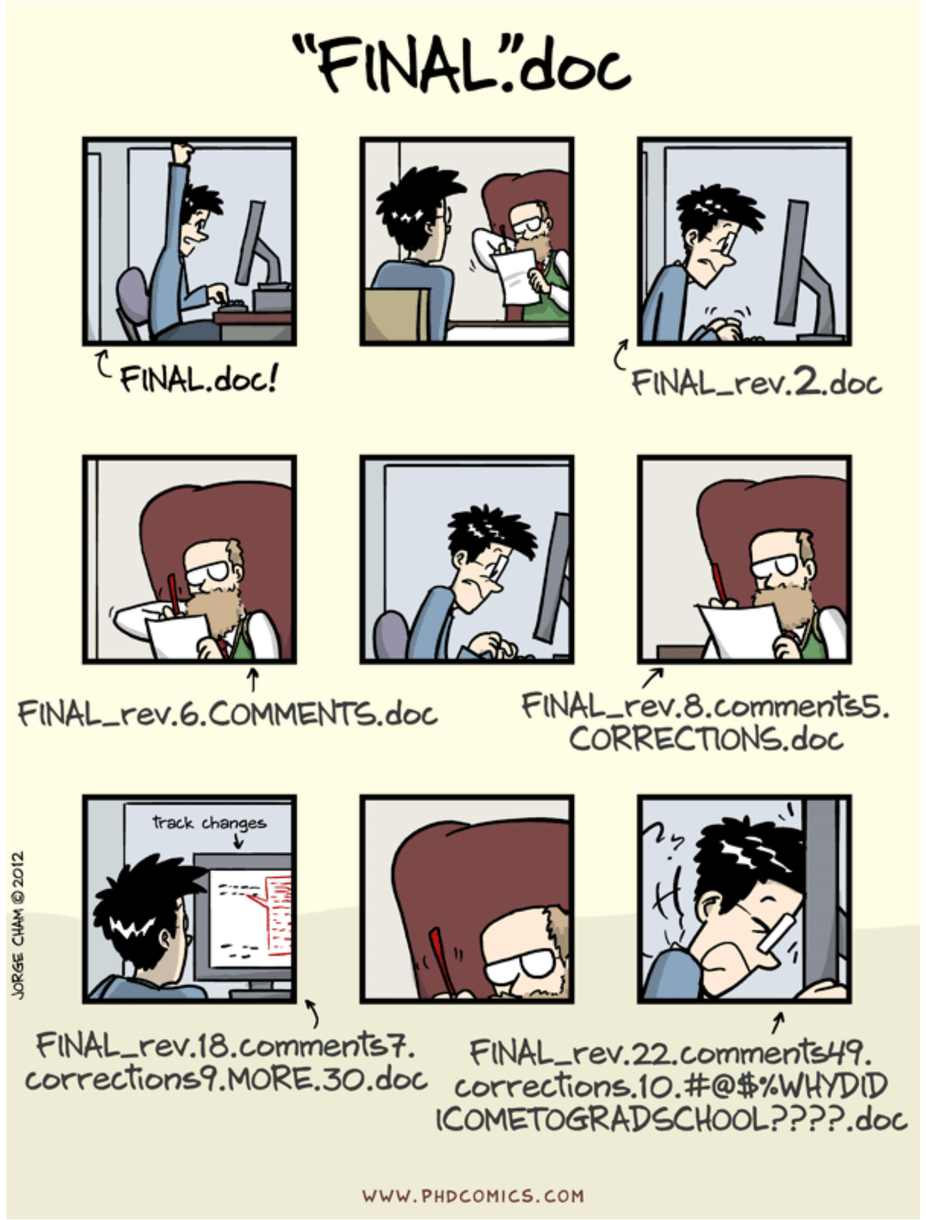
The files we're typically interested in being version controlled are software source code files. However, you can version control almost any type of file

What should we version control?

*"Reproducibility is maximized when **scientists put everything that has been created manually in version control (3.3)**, including programs, original field observations, and the source files for papers."*

Best Practices for Scientific Computing
https://arxiv.org/abs/1210.0530

Have you used version control before?



Version Control System

A version control system (VCS) is a program or set of programs that tracks changes to a collection of files

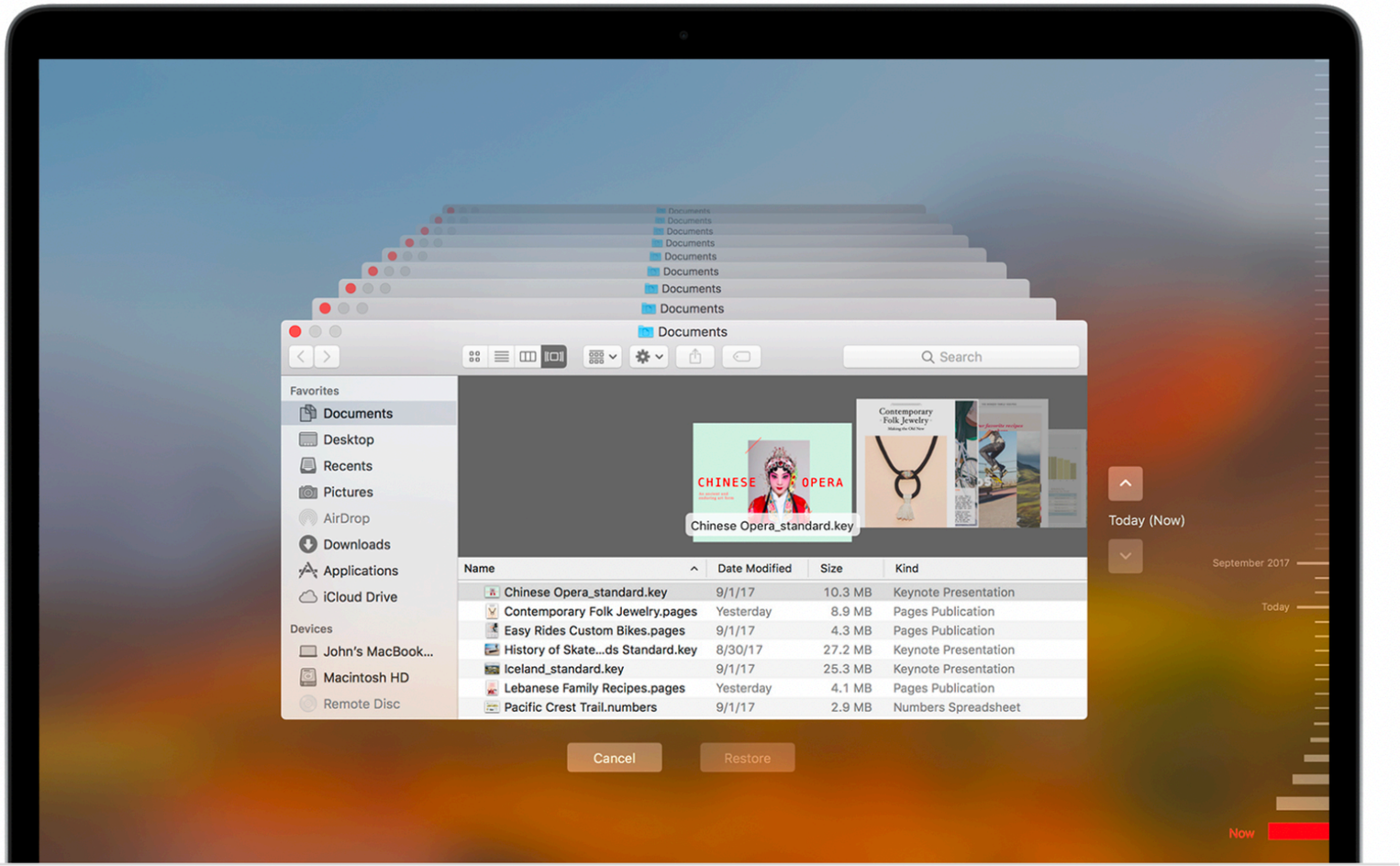
Majors goals of VCS:

- easily recall earlier versions of individual files or the entire project
- allow several team members to work at the same time without affecting each other

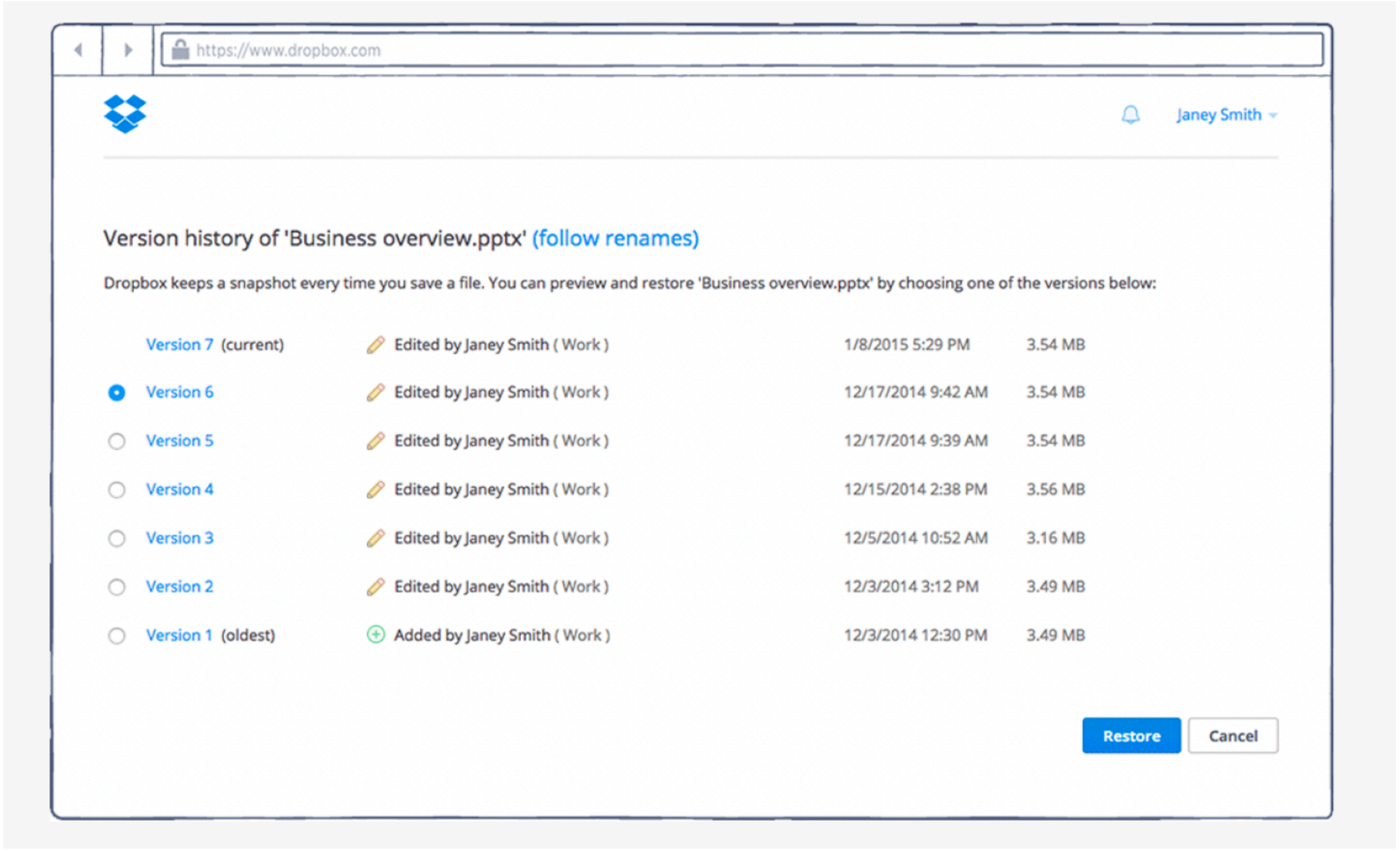
Version control systems keeps track of every modification you make to the files being tracked in a special kind of database.

If a mistake is made, you can then turn back the clock and compare earlier versions of the files to help fix the mistake.

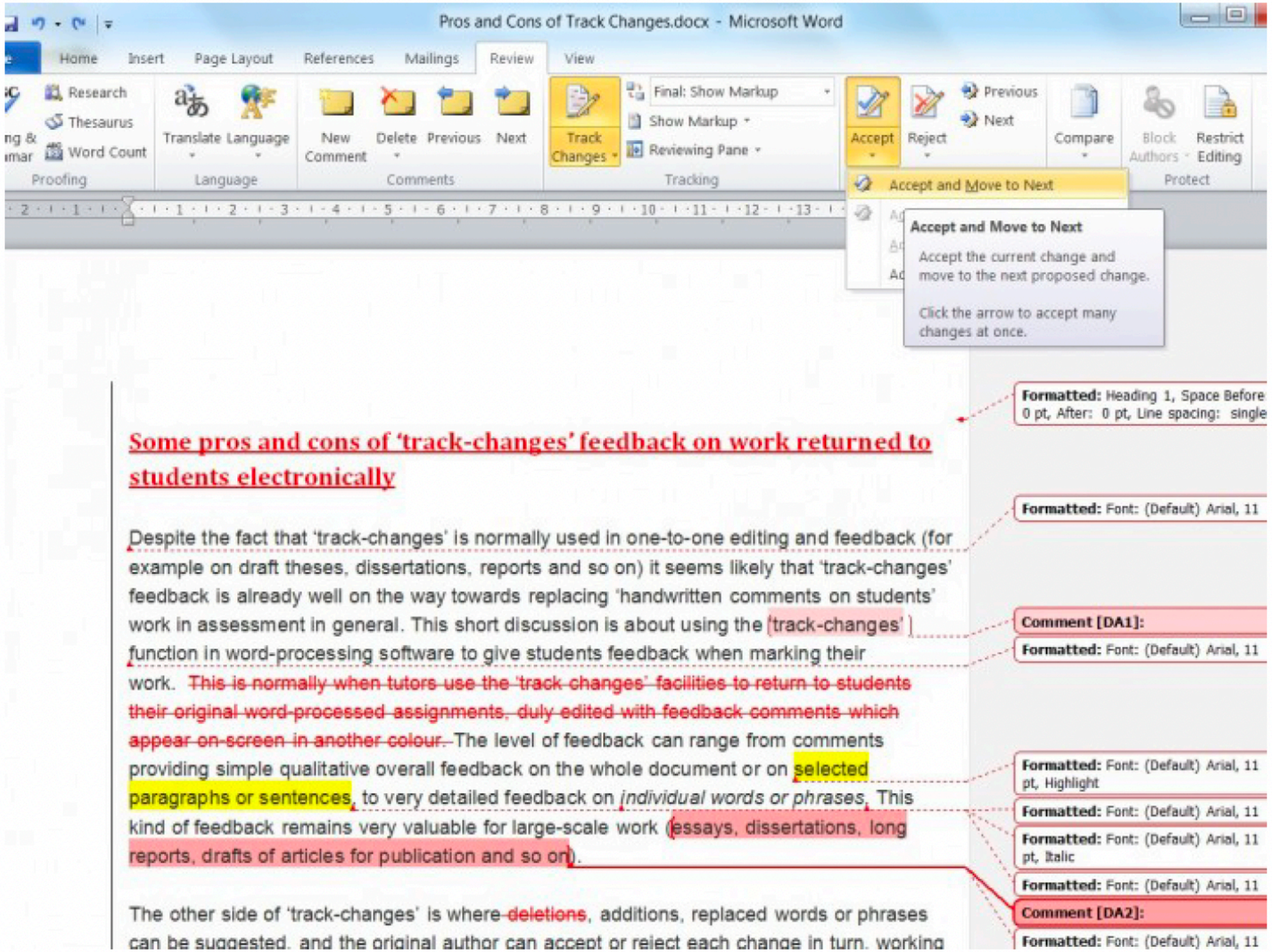
Example: Apple's TimeMachine



Example: Dropbox



Example: Microsoft Word's Track Changes



How does version control work? Record and Playback

Version control systems start with a base version of a document and then simply record the changes you make each step of the way

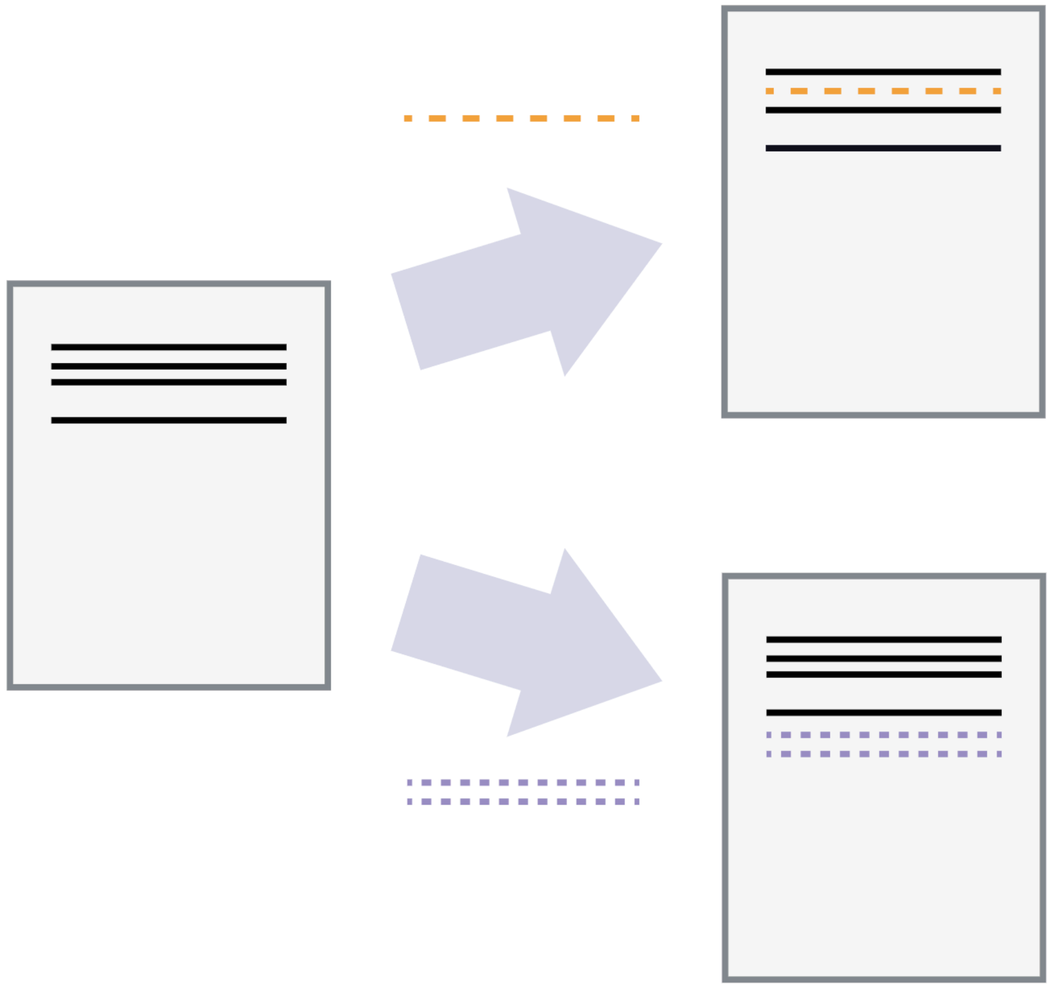
This allows you to rewind to start of the base document and playback each change you made, allowing you to recreate any version of the document in its recorded history

How does version control work? Branching

Independent versions or branches of the same document

Ex. two users can make independent sets of changes on the same document

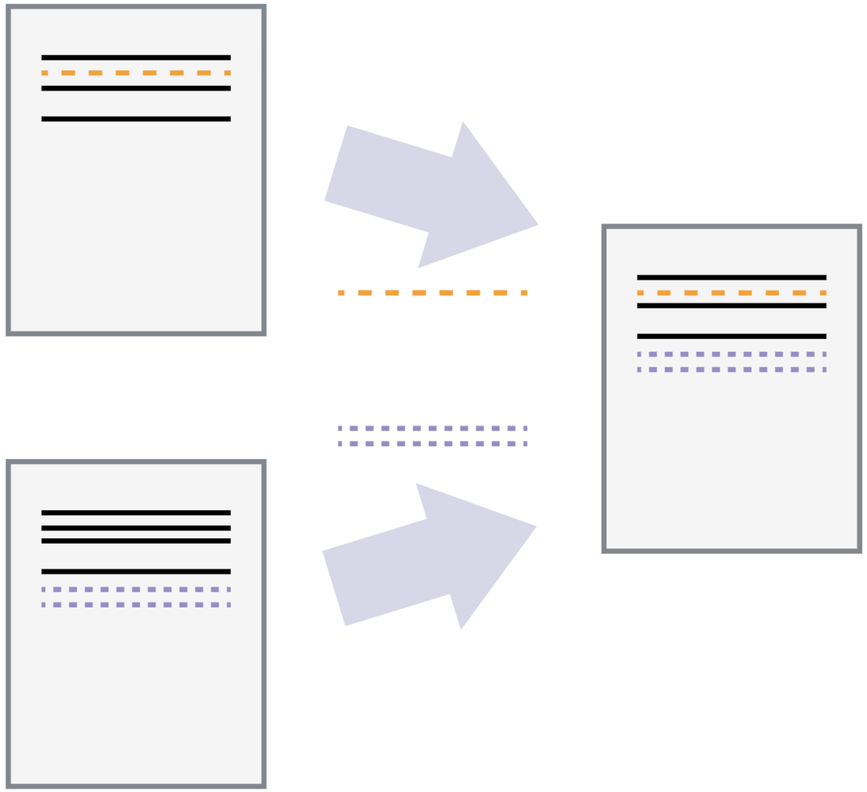
Branching helps facilitate the collaboration amongst multiple contributors



How does version control work? Merging

When a contributor to a document wants to share their changes with their collaborators, they simply need to merge their changes into some common branch of the document shared by the team.

Unless multiple users make changes to the same section of the document - a conflict - you can incorporate two sets of changes into the same base document.



Benefit of using Version Control System

Archiving: You must regularly save the changes you make.

Reproducibility: Creating a history of saved changes allows you to revert selected files or your entire project back to any previous state in its recorded history.

Collaboration: You and a team of contributors can work on a project independently, but share your changes amongst one another as the project takes shape.

Accountability: Compare changes, see who last modified something, and who introduced a problem and when.

Recoverability: Each contributor has their own local, recent copy of the project and its complete history, making it highly unlikely you'll ever lose a significant portion of the project

Local Version Control

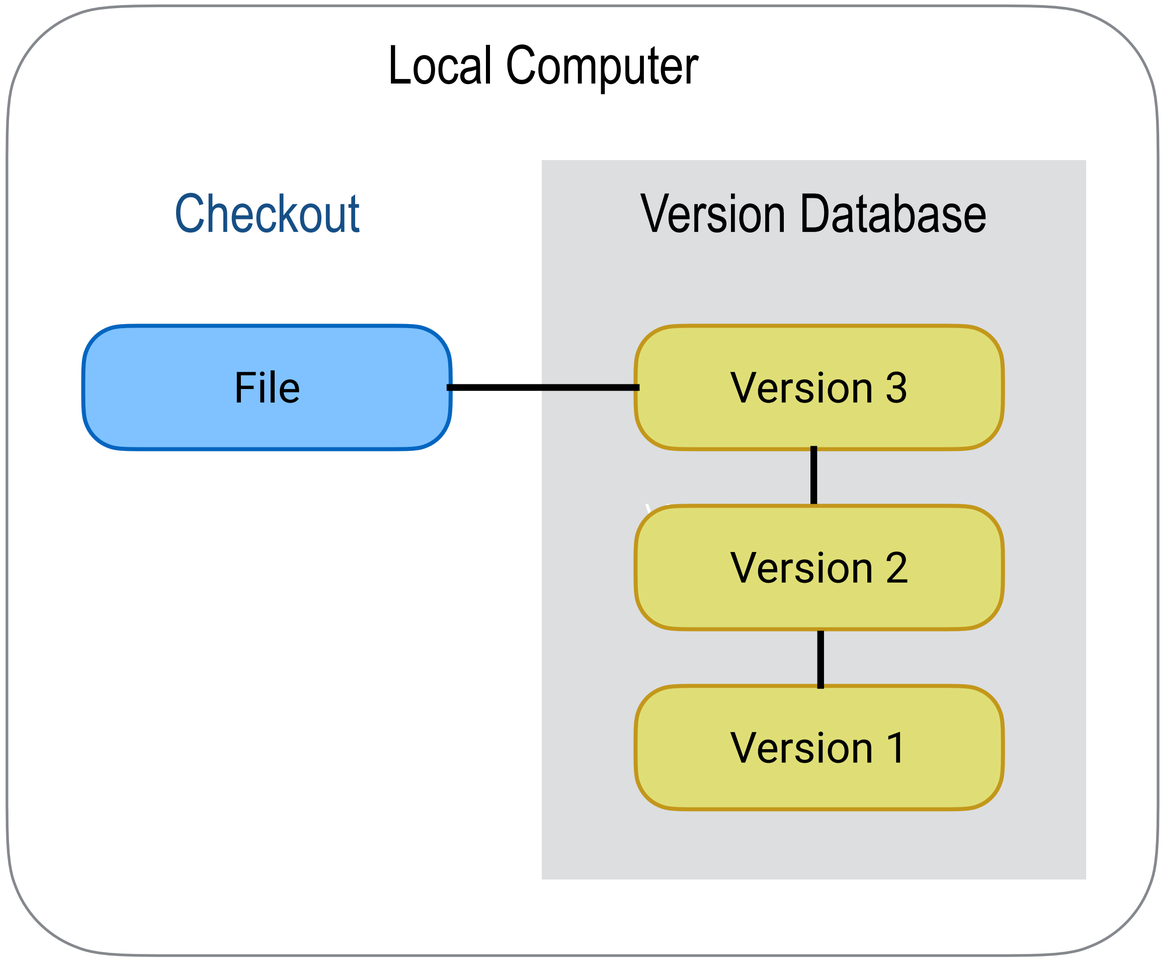
Many people's version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they're clever)

Local VCSs that had a simple database that kept all the changes to files under revision control

Local Version Control

Many people's version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they're clever)

Local VCSs that had a simple database that kept all the changes to files under revision control



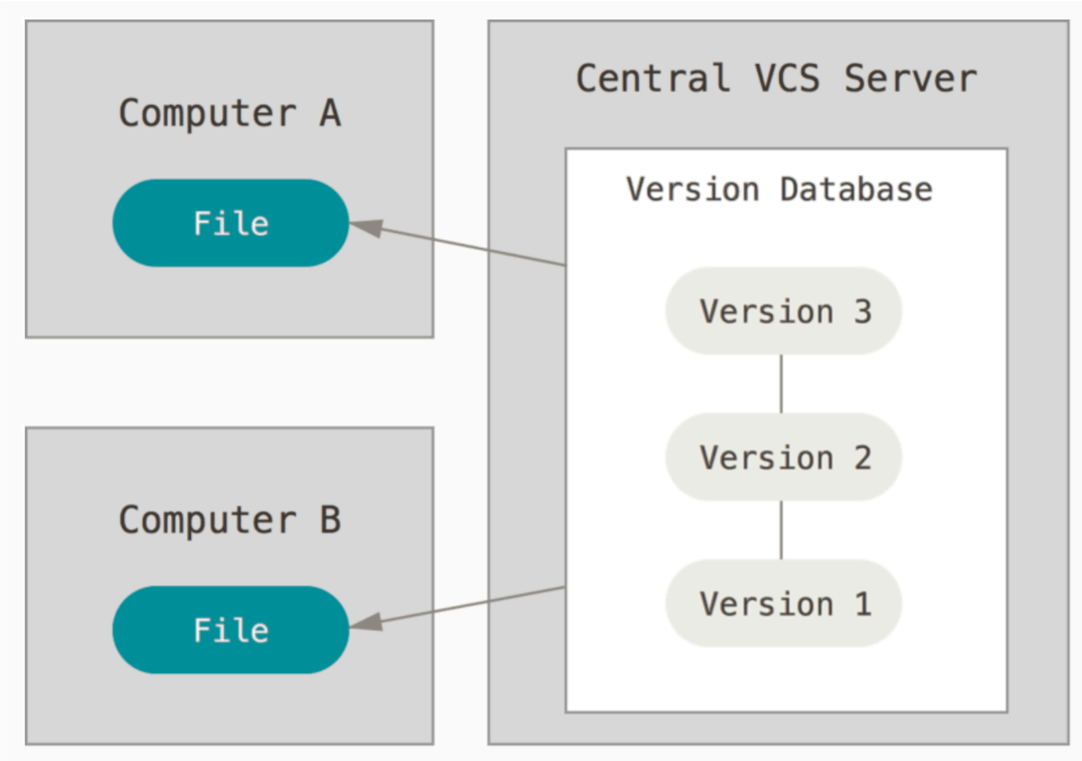
Centralized Version control

These systems have a single server that contains all the versioned files, and a number of clients that check out files from that central place.

Whenever you make a change and want to commit that change, you must send it back to a centralized repository server to be tracked

For many years, this has been the standard for version control.

Example: CVS, Subversion, and Perforce

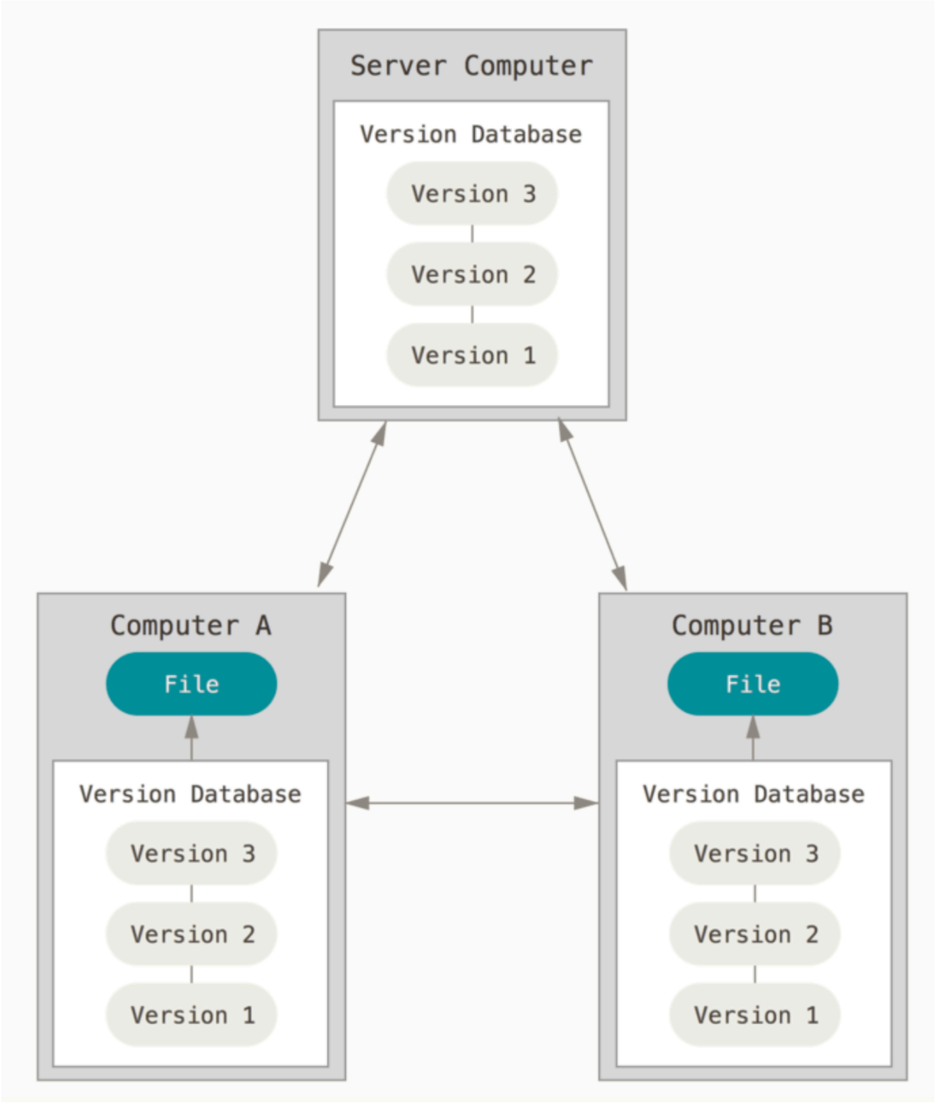


Distributed Version Control

Clients don't just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history

Every clone is really a full backup of all the data.

This allows you to commit changes in a private repository on your local computer, which eliminates the need to be connected to a network where the centralized server is accessible in order to commit a change.

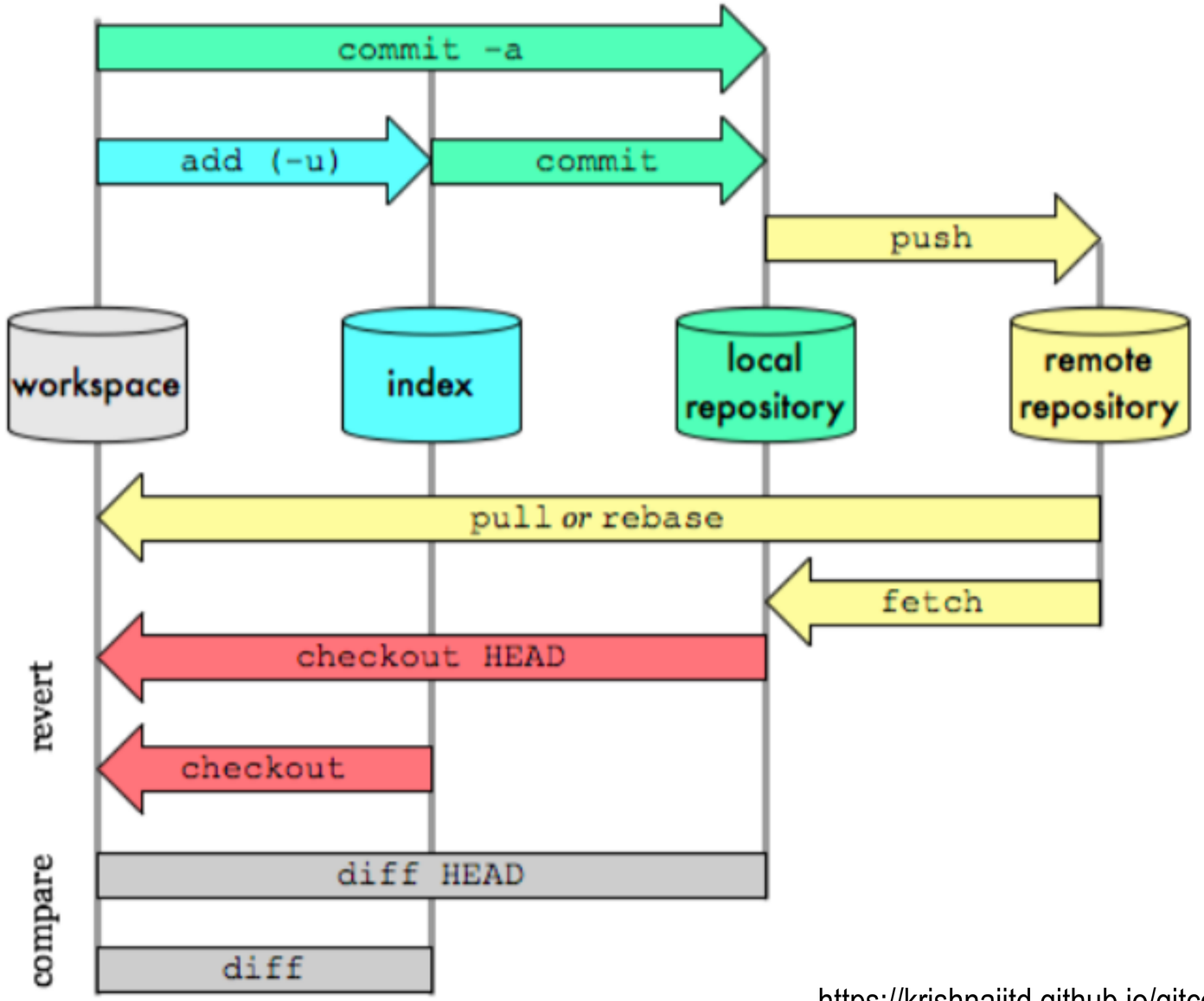


https://git-scm.com

What is git?

- git is a distributed version control system
- git was designed to be simple, fast, and fully-distributed, with support for large projects and nonlinear development workflows
- git was originally created in 2005 by Linus Torvalds and other Linux kernel developers when the free use of the proprietary version control system they had been using for kernel development was revoked by its copyright holder

Git Workflow



https://krishnaiitd.github.io/gitcommands/git-workflow/

Installing git on Linux

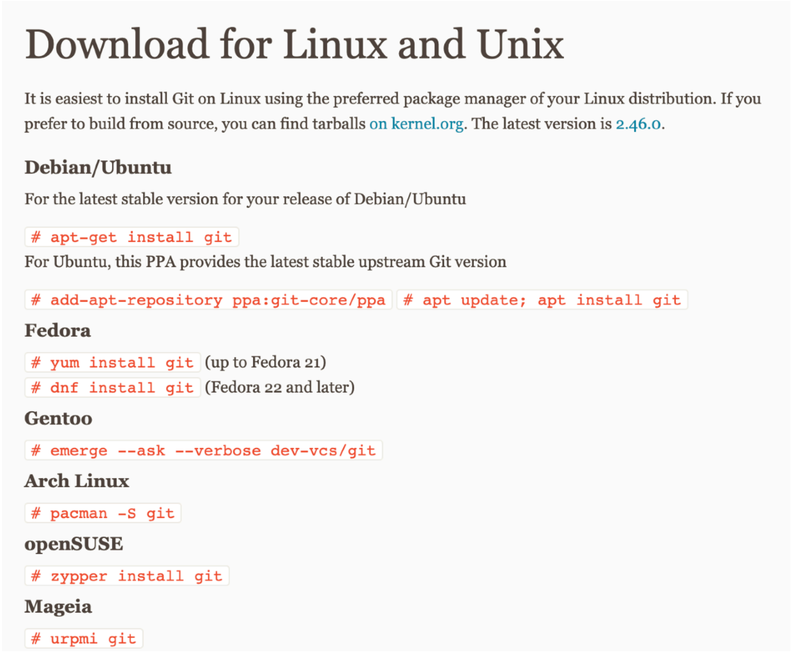
If you want to install git on a Linux-based system, you should be able to do so via your operating system's standard package management tool

For example, on any RPM-based Linux distribution, such as Fedora, RHEL, or CentOS, you can use **dnf**:

```
$ sudo dnf install git-all
```

On any Debian-based distribution, such as Ubuntu, try **apt**:

```
$ sudo apt install git-all
```



Download for Linux and Unix

It is easiest to install Git on Linux using the preferred package manager of your Linux distribution. If you prefer to build from source, you can find tarballs on kernel.org. The latest version is 2.46.0.

Debian/Ubuntu

For the latest stable version for your release of Debian/Ubuntu

`# apt-get install git`

For Ubuntu, this PPA provides the latest stable upstream Git version

`# add-apt-repository ppa:git-core/ppa` `# apt update; apt install git`

Fedora

`# yum install git` (up to Fedora 21)
`# dnf install git` (Fedora 22 and later)

Gentoo

`# emerge --ask --verbose dev-vcs/git`

Arch Linux

`# pacman -S git`

openSUSE

`# zypper install git`

Mageia

`# urpmi git`

Installing git on macOS

There are several ways to install git on your Mac. Probably the easiest way is to install the Xcode Command Line Tools.
On Mavericks (10.9) or above you can do this simply by trying to run `git` from the Terminal the very first time.

```
$ git --version
```

If you don't have it installed already, it will prompt you to install it

If you want a more up to date version, you can also install it via a binary installer. A macOS Git installer is maintained and available for download at the Git website, at https://git-scm.com/download/mac

Installing git on Windows

There are also a few ways to install git under Windows operating systems. However, the official build is available for download on the git website. If you go to

`http://git-scm.com/download/win`

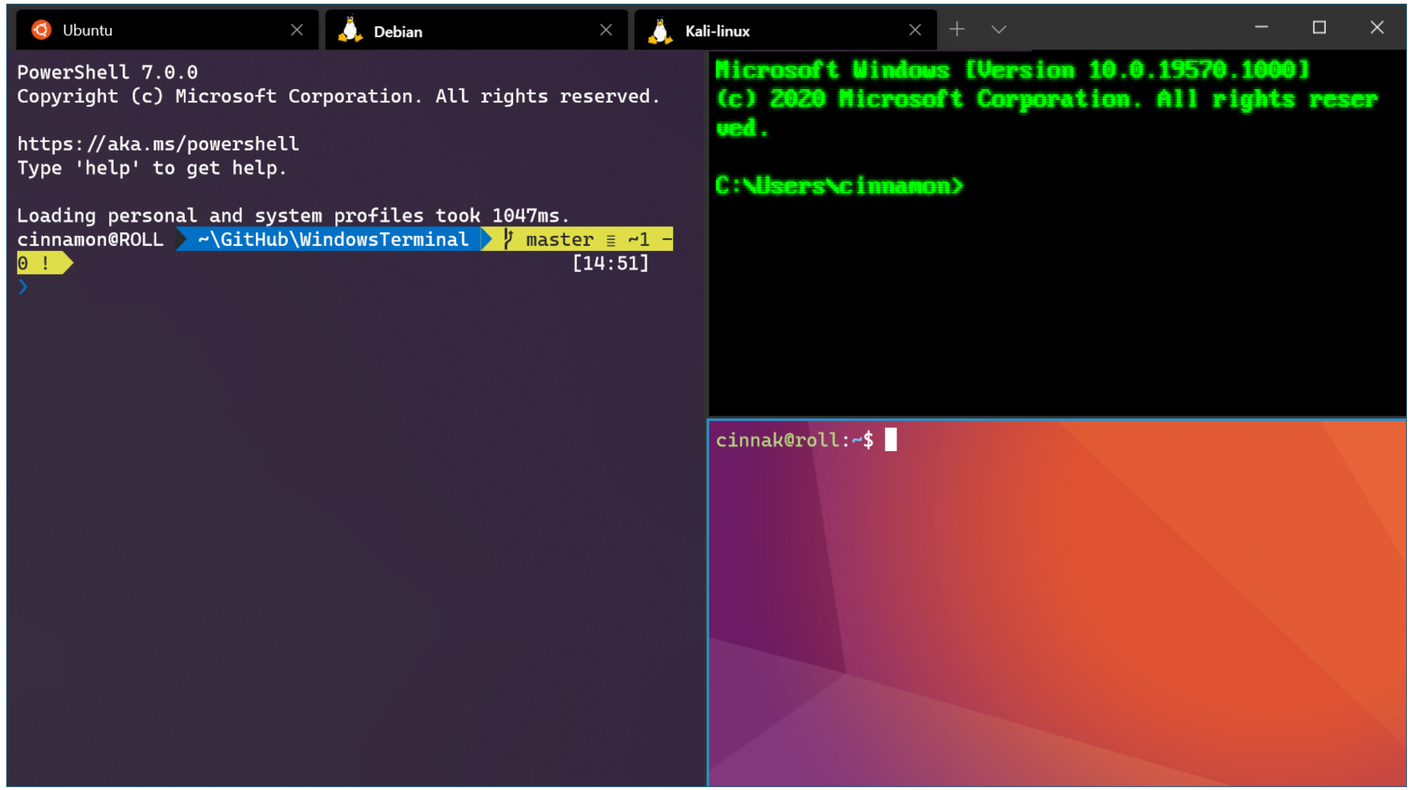
the download should start automatically.

Note: This is a project called “git for Windows”, which is separate from git itself. Please see https://gitforwindows.org for more information.

Windows Subsystem for Linux

Windows Subsystem for Linux (WSL)

lets you run a GNU/Linux environment – including most command-line tools, utilities, and applications – directly on Windows, unmodified, without the overhead of a traditional virtual machine or dual-boot OS setup.



Official documentation:

https://learn.microsoft.com/en-us/windows/wsl/

Installing from source

There is an option to install it from the source

See the official site for more instructions

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Setting up git with git-config

When we start using git for the first time on a new computer, there are always a few things we need to configure before we begin working with git.

Git comes with a tool called `git config` that lets you get and set configuration variables that control all aspects of how Git looks and operates.

Setting up git with git-config

When we start using git for the first time on a new computer, there are always a few things we need to configure before we begin working with git.

Git comes with a tool called `git config` that lets you get and set configuration variables that control all aspects of how Git looks and operates.

```
elham@Elhams-MacBook-Pro ~ % git config
usage: git config [<options>]

Config file location
    --global              use global config file
    --system              use system config file
    --local               use repository config file
    --worktree            use per-worktree config file
    -f, --file <file>     use given config file
    --blob <blob-id>      read config from given blob object

Action
    --get                 get value: name [value-pattern]
    --get-all             get all values: key [value-pattern]
    --get-regexp          get values for regexp: name-regex [value-pattern]
    --get-urlmatch        get value specific for the URL: section[.var] URL
    --replace-all         replace all matching variables: name value [value-pattern]
    --add                 add a new variable: name value
    --unset               remove a variable: name [value-pattern]
    --unset-all           remove all matches: name [value-pattern]
    --rename-section      rename section: old-name new-name
    --remove-section      remove a section: name
    -l, --list            list all
    --fixed-value         use string equality when comparing values to 'value-pattern'
    -e, --edit            open an editor
    --get-color           find the color configured: slot [default]
    --get-colorbool       find the color setting: slot [stdout-is-tty]

Type
    -t, --type <type>     value is given this type
    --bool                value is "true" or "false"
    --int                 value is decimal number
    --bool-or-int         value is --bool or --int
    --bool-or-str         value is --bool or string
    --path                value is a path (file or directory name)
    --expiry-date         value is an expiry date

Other
    -z, --null            terminate values with NUL byte
    --name-only           show variable names only
    --includes            respect include directives on lookup
    --show-origin         show origin of config (file, standard input, blob, command line)
    --show-scope          show scope of config (worktree, local, global, system, command)
    --default <value>     with --get, use default value when missing entry
```

Setting up git with git-config

These configuration variables may be stored in a few different places on your system:

1. **`[path]/etc/gitconf`** file: Contains values applied to every user on the system and all their repositories.
   If you pass the option `--system` to `git config`, it reads and writes from this file specifically

2. **`~/.gitconfig`** or **`~/.config/git/config`** file: Values specific personally to you, the user. Contains configuration variables of all of the repositories you work with on your system
   If you pass the `--global` option, Git reads and writes to this file specifically

3. **`~/path/to/your/repo/ .git/config`** file: Specific to that single repository.
   If you pass the `--local` option, Git reads and writes to this file specifically

Setting up your Identity with git-config

The first thing you should do when you install Git is to set your user name and email address

```
$ git config --global user.name “Elham E Khoda"
$ git config --global user.email elhamekhoda@gmail.com
```

These configuration variables are important because this information is included in every change (`git commit`) you make in order to provide a history of who made what changes

Setting up default branch with git config

By default Git will create a branch called `master` when you create a new repository with git init.

From Git version 2.28 onwards, you can set a different name for the initial branch.

To set `main` as the default branch name do:

```
git config --global init.defaultBranch main
```

Setting up colors with git config

In addition to your identity and default text editor, you'll also probably want to make sure that you turn on color highlighting in git's user interface.

You can do so with the following command:

```
git config --global color.ui "auto"
```

Checking your settings

To check the configuration variables you've set in your git config, you can run the command

```
git config --list
```

to list all of the variable values.

If you only need to check one variable in the configuration, you can always replace the `--list` option with the name of the configuration variable itself.

For example, if you wanted to check the value of user.email you've set, you'd run the command:

```
git config user.email
```

Getting git help

If you ever need help while using git, you can access the comprehensive manual page (manpage) for any git command via the git help command:

```
$ git help <command>
$ git <command> --help
$ man git-<command>
```

For example:

```
$ git help config
```

If you only need a quick reference to the options available for a given git command, you can use the `-h` option appended to the command itself:

```
$ git <command> -h
```

Create Your First git Repository

If you have a project on your local system that is not currently under version control and you want to start tracking it git, you can create a git repository for that project by first going to that project's directory

```
$ cd /path/to/your/project
```

and then typing the command:

```
$ git init
```

This creates a new subdirectory named `.git` in the project directory where the repository will live.

If you ever want to stop tracking the project with git, you simply need to delete this subdirectory.

Initialize your repository with git init

```
[elham@login02 code_demo]$
[elham@login02 code_demo]$ ls
[elham@login02 code_demo]$ mkdir planets
[elham@login02 code_demo]$ ls -la
total 19
drwxr-xr-x 3 elham use300  3 Aug  6 16:45 .
drwxr-x--- 4 elham use300 13 Aug  6 16:44 ..
drwxr-xr-x 2 elham use300  2 Aug  6 16:45 planets
[elham@login02 code_demo]$
[elham@login02 code_demo]$ git status
fatal: not a git repository (or any parent up to mount point /home)
Stopping at filesystem boundary (GIT_DISCOVERY_ACROSS_FILESYSTEM not set).
[elham@login02 code_demo]$ cd planets/
[elham@login02 planets]$ git status
fatal: not a git repository (or any parent up to mount point /home)
Stopping at filesystem boundary (GIT_DISCOVERY_ACROSS_FILESYSTEM not set).
[elham@login02 planets]$
[elham@login02 planets]$ git init
Initialized empty Git repository in /home/elham/code_demo/planets/.git/
[elham@login02 planets]$ ls -la
total 19
drwxr-xr-x 3 elham use300  3 Aug  6 16:46 .
drwxr-xr-x 3 elham use300  3 Aug  6 16:45 ..
drwxr-xr-x 7 elham use300 10 Aug  6 16:46 .git
[elham@login02 planets]$ git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
[elham@login02 planets]$
```

State of your repository with git status

Each file in your repository's working directory may take on one of a few different states. The main tool you use to determine which files are in which state is with the command:

```
$ git status
```

State of your repository with git status

Each file in your repository's working directory may take on one of a few different states. The main tool you use to determine which files are in which state is with the command:

```
$ git status
```

What do you see?

State of your repository with git status

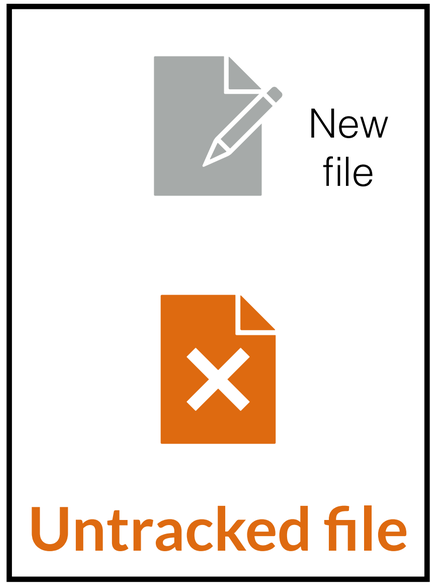
Each file in your repository's working directory may take on one of a few different states. The main tool you use to determine which files are in which state is with the command:

```
$ git status
```

What do you see?

```
[elham@login02 planets]$ git status
On branch main

No commits yet

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

Tracked and Untracked files



There are two primary file states you will find in your repository:

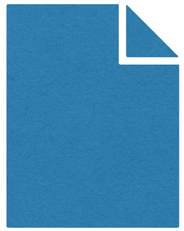
1. **Tracked files** are all of the files that git already knows about and has under version control

2. **Untacked files** are everything else — any files in your project's working directory not yet under version control.

States of tracked files

Unmodified / committed

When a file gets committed, the changes made to it are safely stored in a snapshot in the Git directory



All files that are staged will be part of the next snapshot we take

Staged

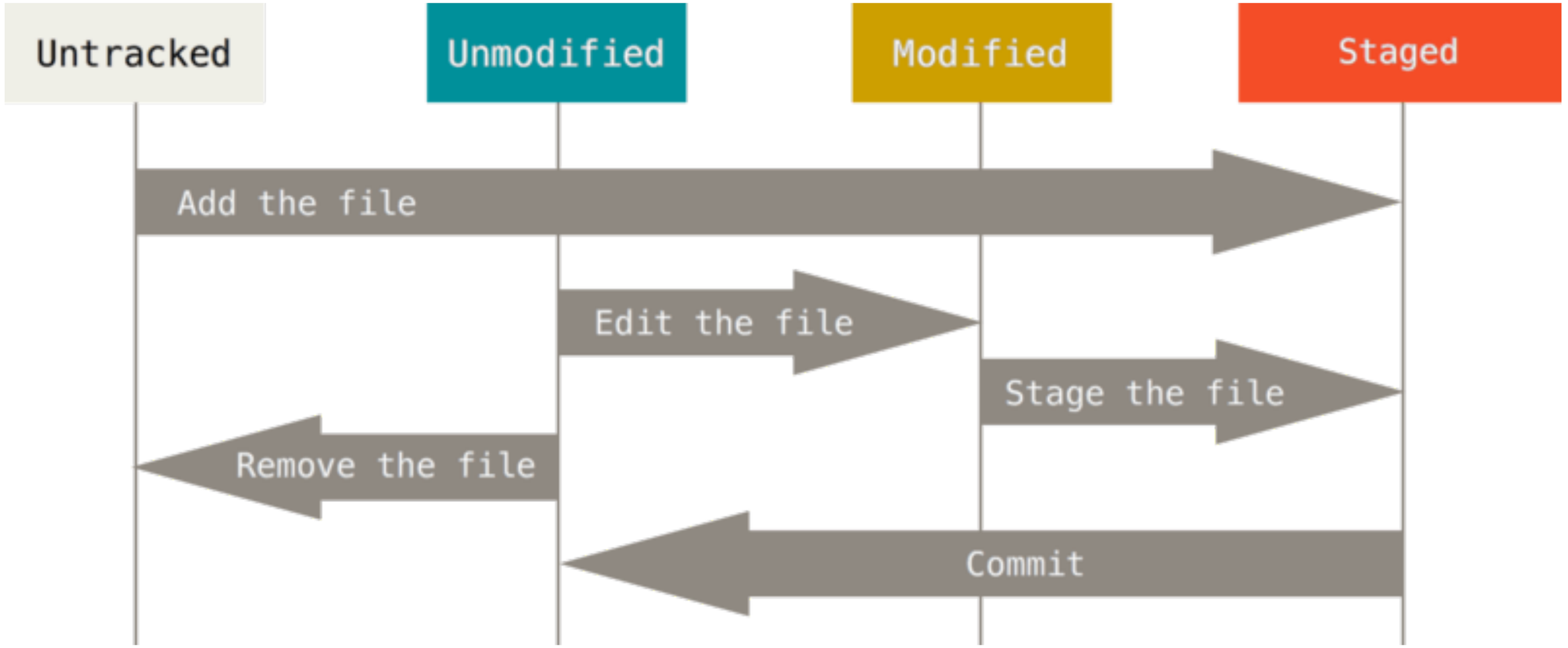
In other words, the changes to those files are ready to be committed to the project

Git notices anytime we modify our files. But won't store any changes until we add them to the staging area.

Modified

- Adding
- Deleting
- Modifying

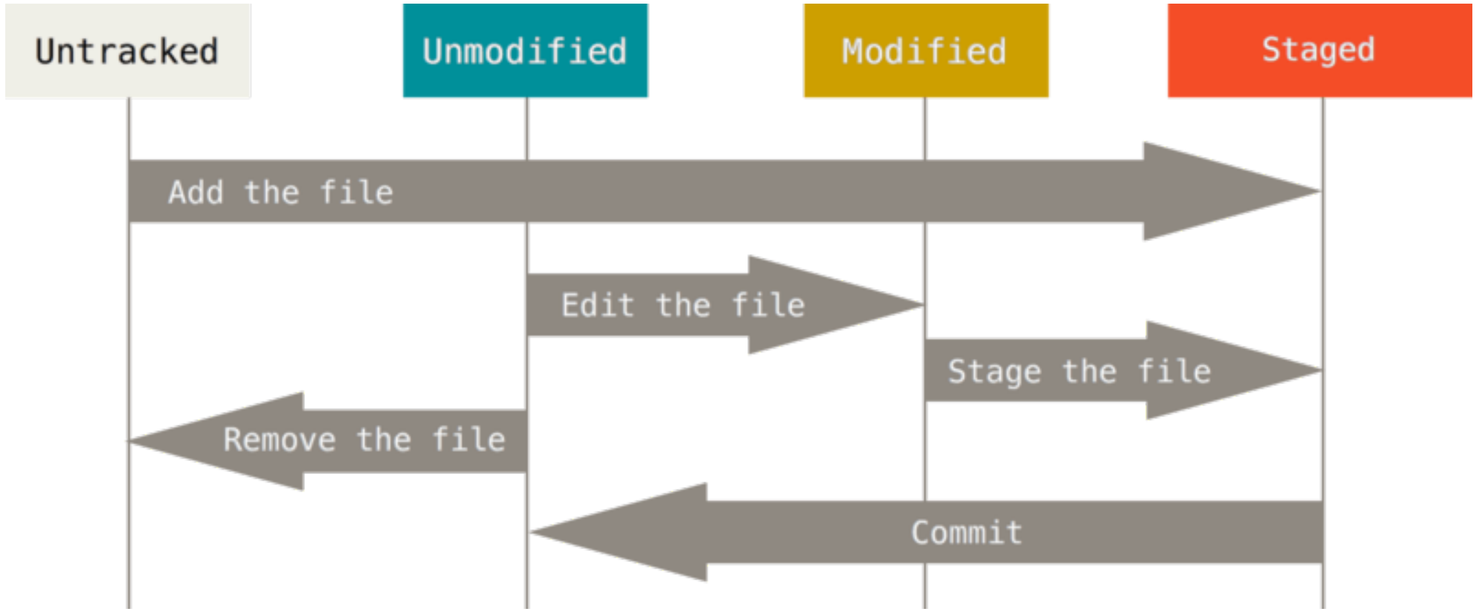
Recording changes to the repository



Lets create a new file

```
[elham@login02 planets]$
[elham@login02 planets]$
[elham@login02 planets]$ echo "Cold and dry, but everything is my favorite color." > mars.txt
[elham@login02 planets]$ ls -lahtr
total 19K
drwxr-xr-x 3 elham use300  3 Aug  6 16:45 ..
drwxr-xr-x 7 elham use300 10 Aug  6 16:46 .git
drwxr-xr-x 3 elham use300  4 Aug  6 18:43 .
-rw-r--r-- 1 elham use300 51 Aug  6 18:44 mars.txt
[elham@login02 planets]$ git status
On branch main

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
[elham@login02 planets]$
```

Add files to your repository with git add



To begin tracking a new file in your project's working directory, you must first stage the new file to be committed to the repository with the command:

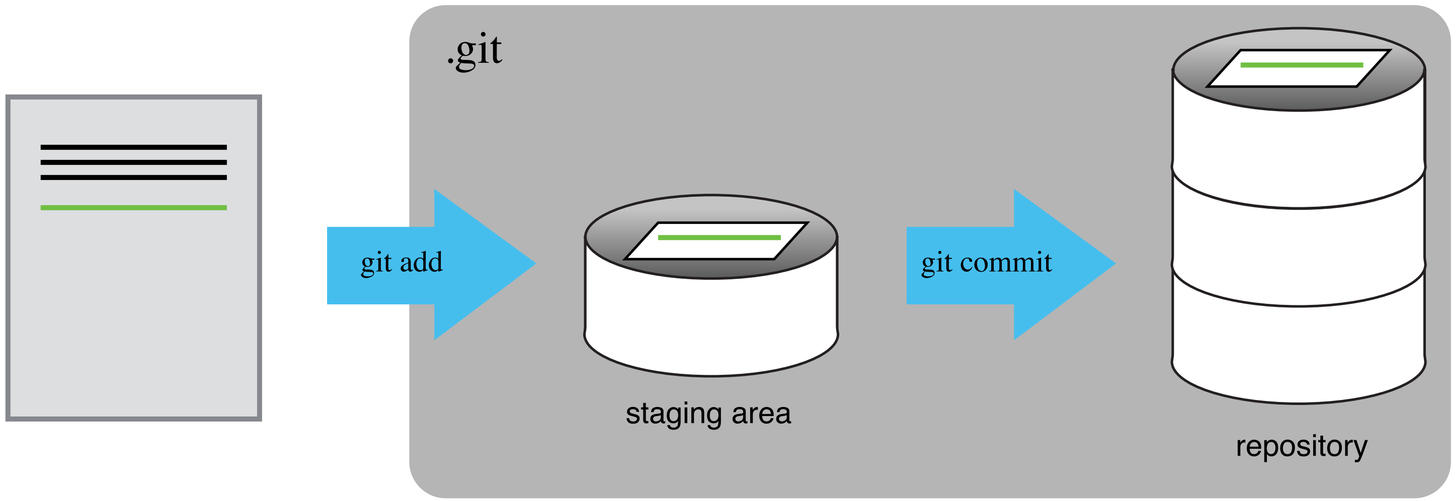
```
git add <path/to/file/filename>
```

If you run git status again, you should see that the newly added file is now tracked and staged to be committed

Add files to your repository with git add

```
[elham@login02 planets]$ git status
On branch main

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
[elham@login02 planets]$
[elham@login02 planets]$
[elham@login02 planets]$ git add mars.txt
[elham@login02 planets]$ git status
On branch main

No commits yet

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

[elham@login02 planets]$
```

Committing changes with git commit



Changes to files in your repository are tracked through commits, which you make with the

```
git commit
```

Prior to a commit, you need to stage the files whose changes you want to commit using the git add command

Writing a commit message

Each commit requires a commit message. They provide the context about why a change was made. This may be to inform other collaborators working on the project about the change or, more often, to remind yourself why you made the change.

How you choose to format your commit messages is up to you and your team. But if you need some guidance, here are seven rules to writing a great git commit message:

https://chris.beams.io/posts/git-commit

Committing changes with git commit

```
git commit -m <your commit message>
```

```
[elham@login02 planets]$ git status
On branch main

No commits yet

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

[elham@login02 planets]$ git commit -m "Added my initial description of the planet Mars in mars.txt"
[main (root-commit) 6d58043] Added my initial description of the planet Mars in mars.txt
 1 file changed, 1 insertion(+)
 create mode 100644 mars.txt
[elham@login02 planets]$ git status
On branch main
nothing to commit, working tree clean
[elham@login02 planets]$ git log
commit 6d580438248b251f6060bcd25e9f5aa153a6c242 (HEAD -> main)
Author: Elham E Khoda <elhamekhoda@gmail.com>
Date:   Tue Aug 6 19:35:45 2024 -0700

    Added my initial description of the planet Mars in mars.txt
[elham@login02 planets]$
```

View commit history with git log

The power of git is that it tracks the changes to files in your project over time. You can use the

```
git log
```

command to view that history

In the standard output of the git log command, you will see
- the commit ID of each commit
- the author who made it
- the date it was committed, and
- the commit message explaining the reason for the change(s)

By default, the git log command will list commit history in reverse chronological order.

Lets add some more file and more changes

Now it's your turn!

1. Add two more files: `earth.txt` and `pluto.txt`
2. Add a line to each of these files
3. Add another line to `mars.txt`

Lets add some more file and more changes

```
[elham@login02 planets]$
[elham@login02 planets]$
[elham@login02 planets]$ git status
On branch main
nothing to commit, working tree clean
[elham@login02 planets]$ echo "A pale blue dot, but it is home." > earth.txt
[elham@login02 planets]$ echo "Not a planet." > pluto.txt
[elham@login02 planets]$ echo "Well, not the sunsets -- they are blue" >> mars.txt
[elham@login02 planets]$ 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:   mars.txt

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

no changes added to commit (use "git add" and/or "git commit -a")
[elham@login02 planets]$
```

Staging New files and committing them

```
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:   mars.txt

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

no changes added to commit (use "git add" and/or "git commit -a")
[elham@login02 planets]$ git add earth.txt pluto.txt
[elham@login02 planets]$ git commit -m "Added my initial descriptions of earth in earth.txt; also added my initial
descriptions of pluto in pluto.txt, even though it is only a minor planet now."
[main fe45fcf] Added my initial descriptions of earth in earth.txt; also added my initial descriptions of pluto in
pluto.txt, even though it is only a minor planet now.
 2 files changed, 2 insertions(+)
 create mode 100644 earth.txt
 create mode 100644 pluto.txt
[elham@login02 planets]$ 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:   mars.txt

no changes added to commit (use "git add" and/or "git commit -a")
[elham@login02 planets]$ git log
commit fe45fcff646078a1ea9f8fcb72dc5223952348bd (HEAD -> main)
Author: Elham E Khoda <elhamekhoda@gmail.com>
Date:   Tue Aug 6 21:53:55 2024 -0700

    Added my initial descriptions of earth in earth.txt; also added my initial descriptions of pluto in pluto.txt,
even though it is only a minor planet now.

commit 6d580438248b251f6060bcd25e9f5aa153a6c242
Author: Elham E Khoda <elhamekhoda@gmail.com>
Date:   Tue Aug 6 19:35:45 2024 -0700

    Added my initial description of the planet Mars in mars.txt
[elham@login02 planets]$
```

Comparing differences with git diff

While git log provides you with information about a commit – who made it, when they made it, and why they made it — sometimes looking at the specific changes made to the files themselves is what's more important.

You can use the

```
git diff
```

command to see the changes to any modified files in your working directory before you stage them to be committed.

Comparing differences with git diff

```
[elham@login02 planets]$ 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:   mars.txt

no changes added to commit (use "git add" and/or "git commit -a")
[elham@login02 planets]$ git diff
diff --git a/mars.txt b/mars.txt
index 077071c..c1e6f83 100644
--- a/mars.txt
+++ b/mars.txt
@@ -1 +1,2 @@
 Cold and dry, but everything is my favorite color.
+Well, not the sunsets -- they are blue
[elham@login02 planets]$
```

Breaking down git diff

1. The first line tells us that Git is producing output similar to the Unix diff command comparing the old and new versions of the file

2. The second line tells exactly which versions of the file Git is comparing

3. The third and fourth lines once again show the name of the file being changed

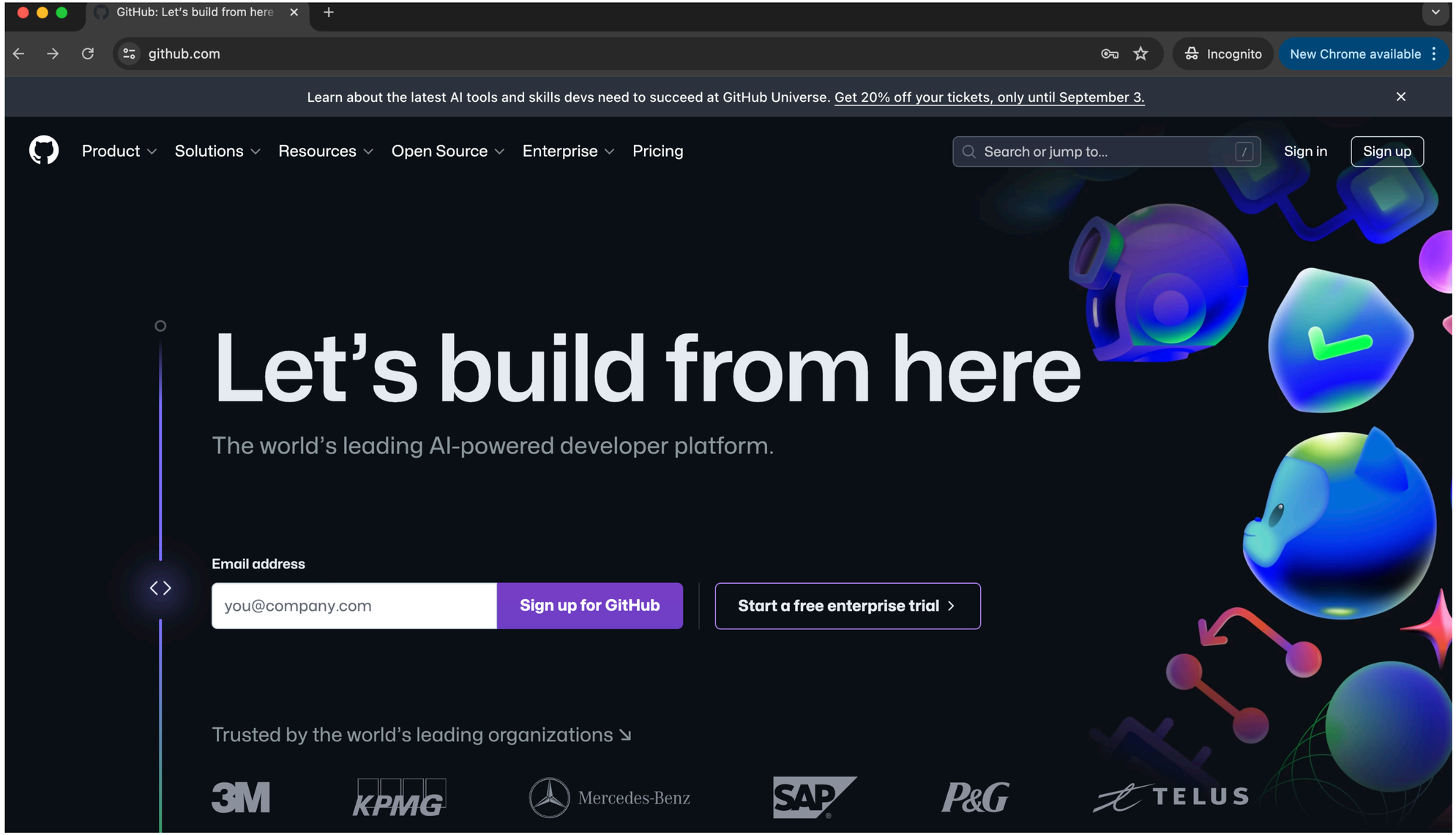
4. The remaining lines are the most interesting, they show us the actual differences and the lines on which they occur. In particular, the + marker in the first column shows where we added a line.

Let’s commit after reviewing the changes

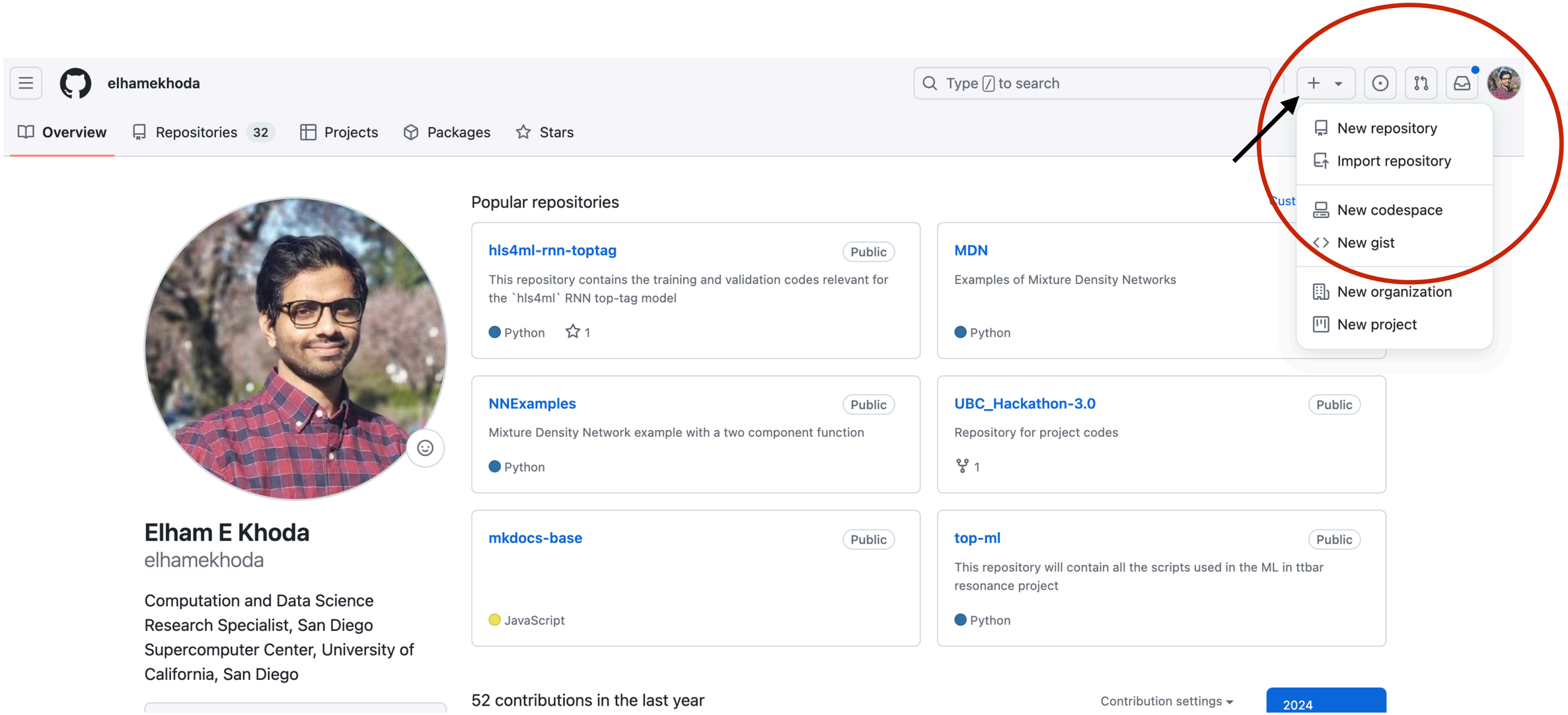
```
[elham@login02 planets]$ git diff
diff --git a/mars.txt b/mars.txt
index 077071c..c1e6f83 100644
--- a/mars.txt
+++ b/mars.txt
@@ -1 +1,2 @@
 Cold and dry, but everything is my favorite color.
+Well, not the sunsets -- they are blue
[elham@login02 planets]$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   mars.txt

no changes added to commit (use "git add" and/or "git commit -a")
[elham@login02 planets]$
[elham@login02 planets]$ git add mars.txt
[elham@login02 planets]$ git commit -m "I forgot there is at least one color on Mars, that is not my favorite"
[main 9c819e6] I forgot there is at least one color on Mars, that is not my favorite
 1 file changed, 1 insertion(+)
[elham@login02 planets]$ git diff
[elham@login02 planets]$ git status
On branch main
nothing to commit, working tree clean
[elham@login02 planets]$
```

https://github.com/

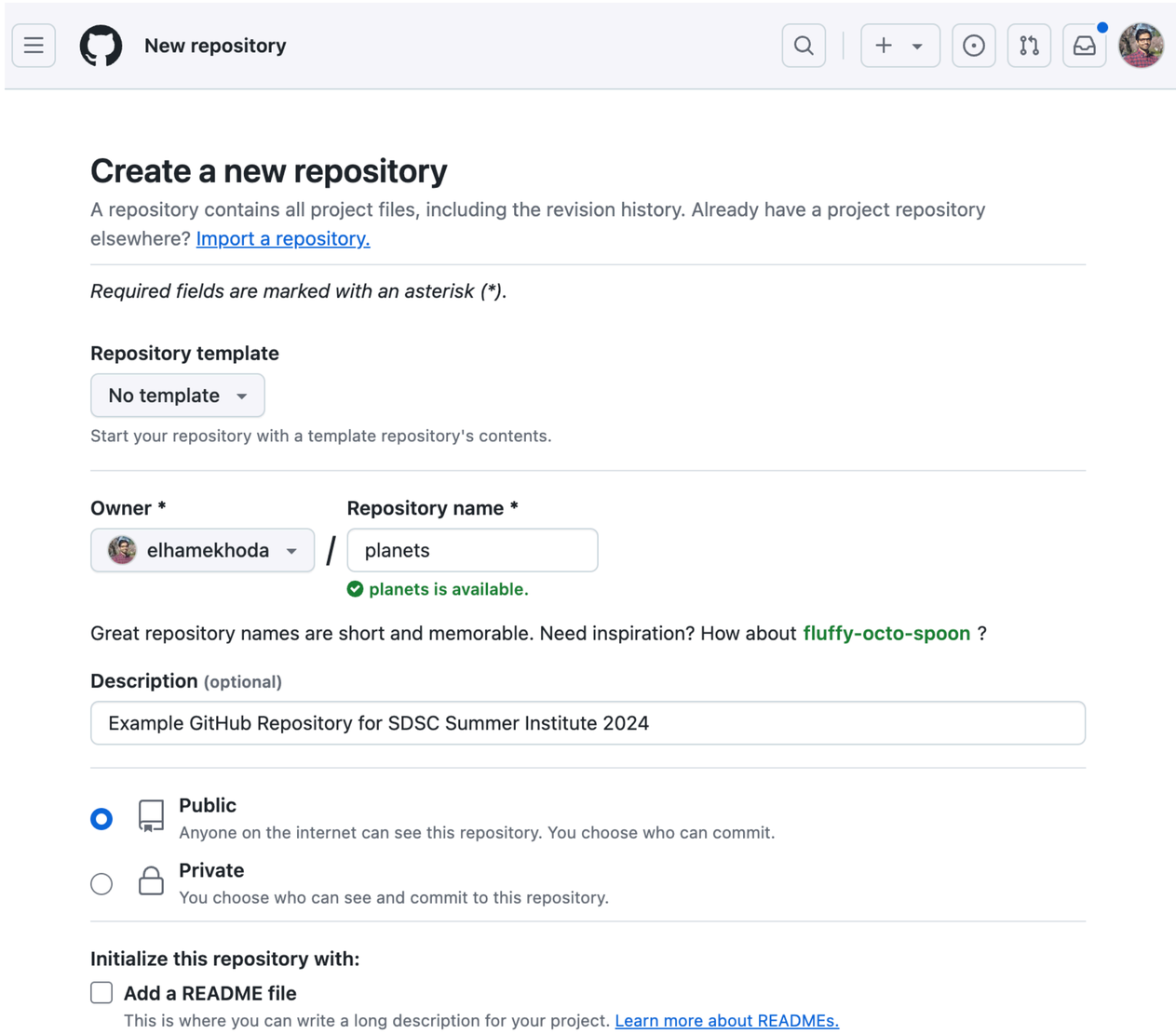
If you are new to GitHub: Sign up



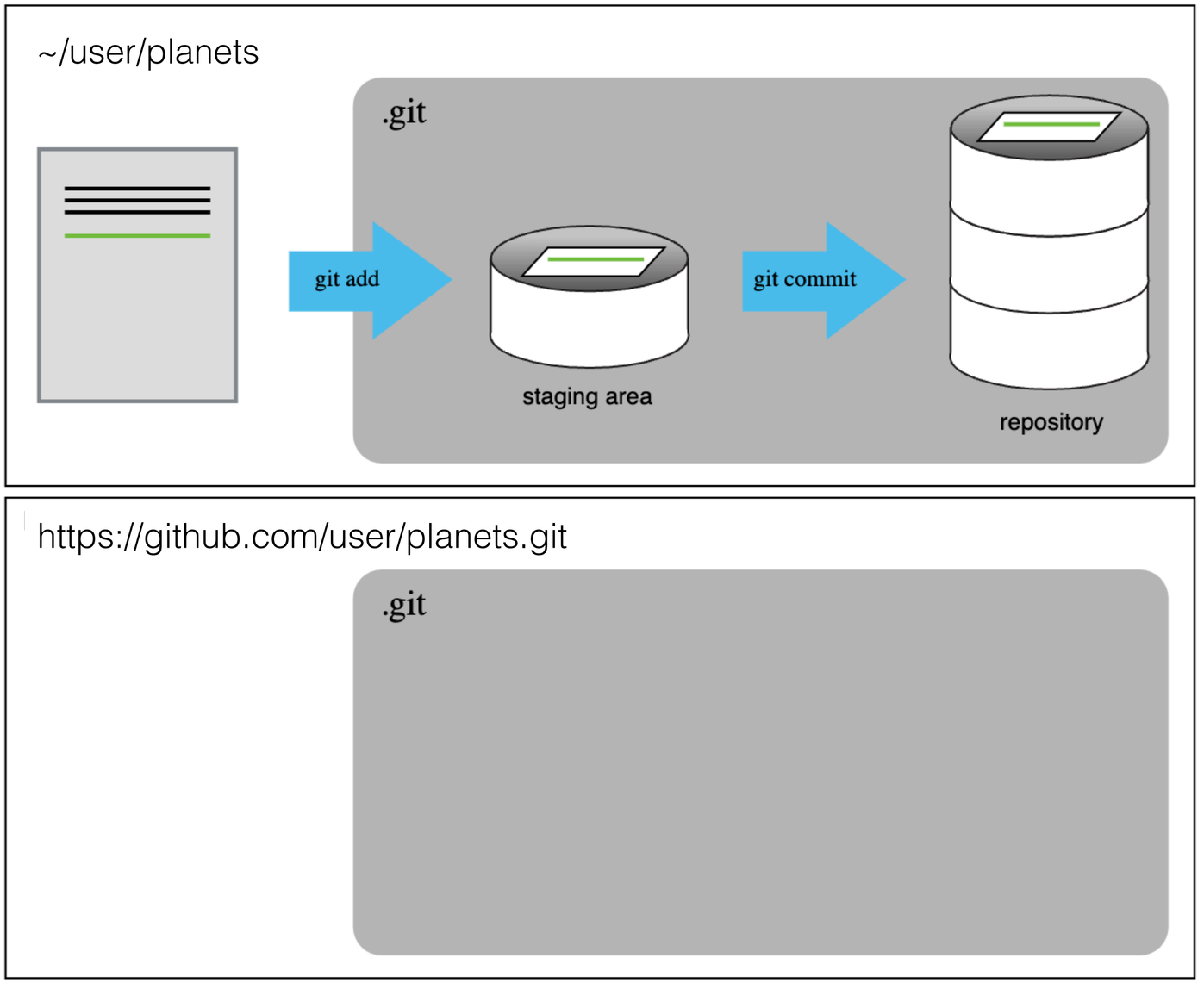
Your GitHub profile



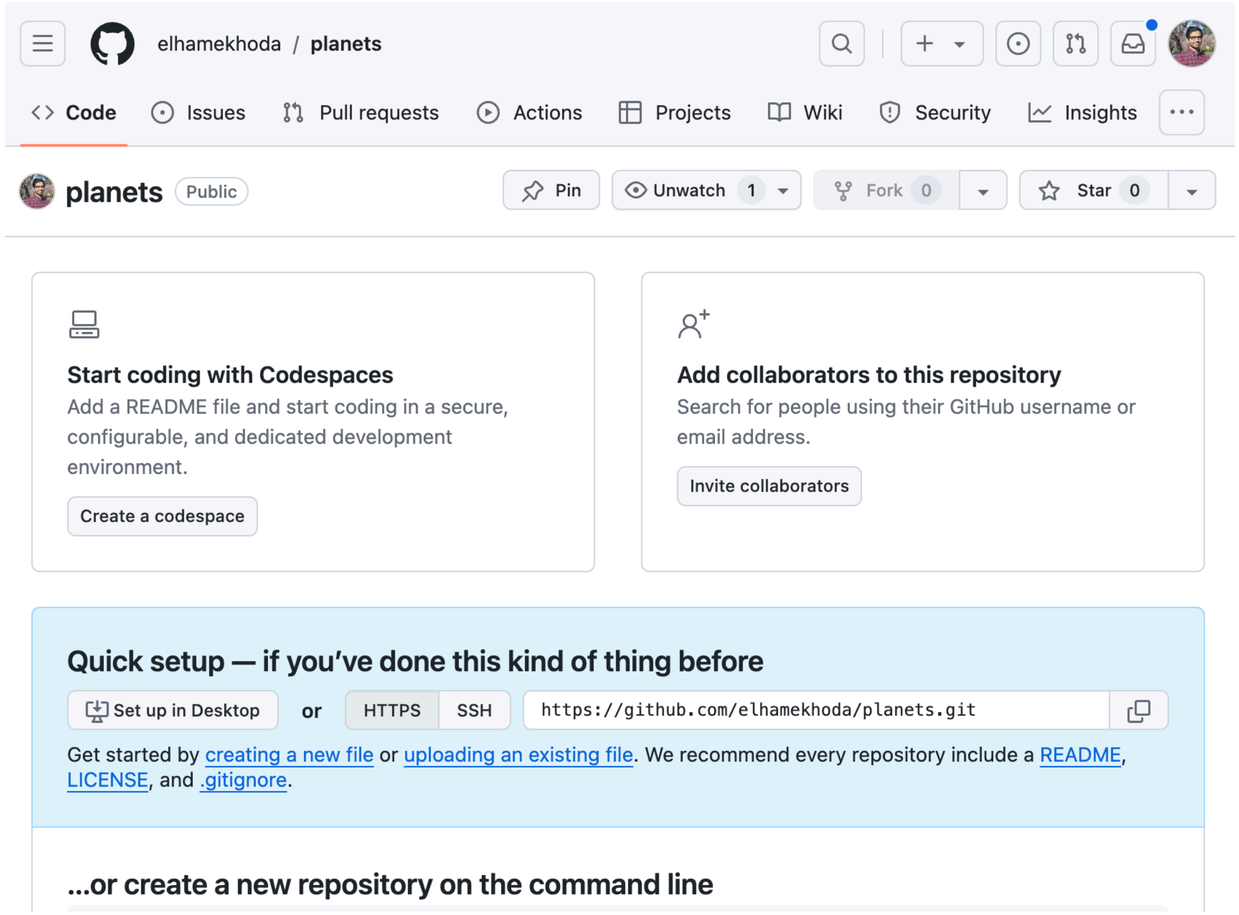
Create a new GitHub repository

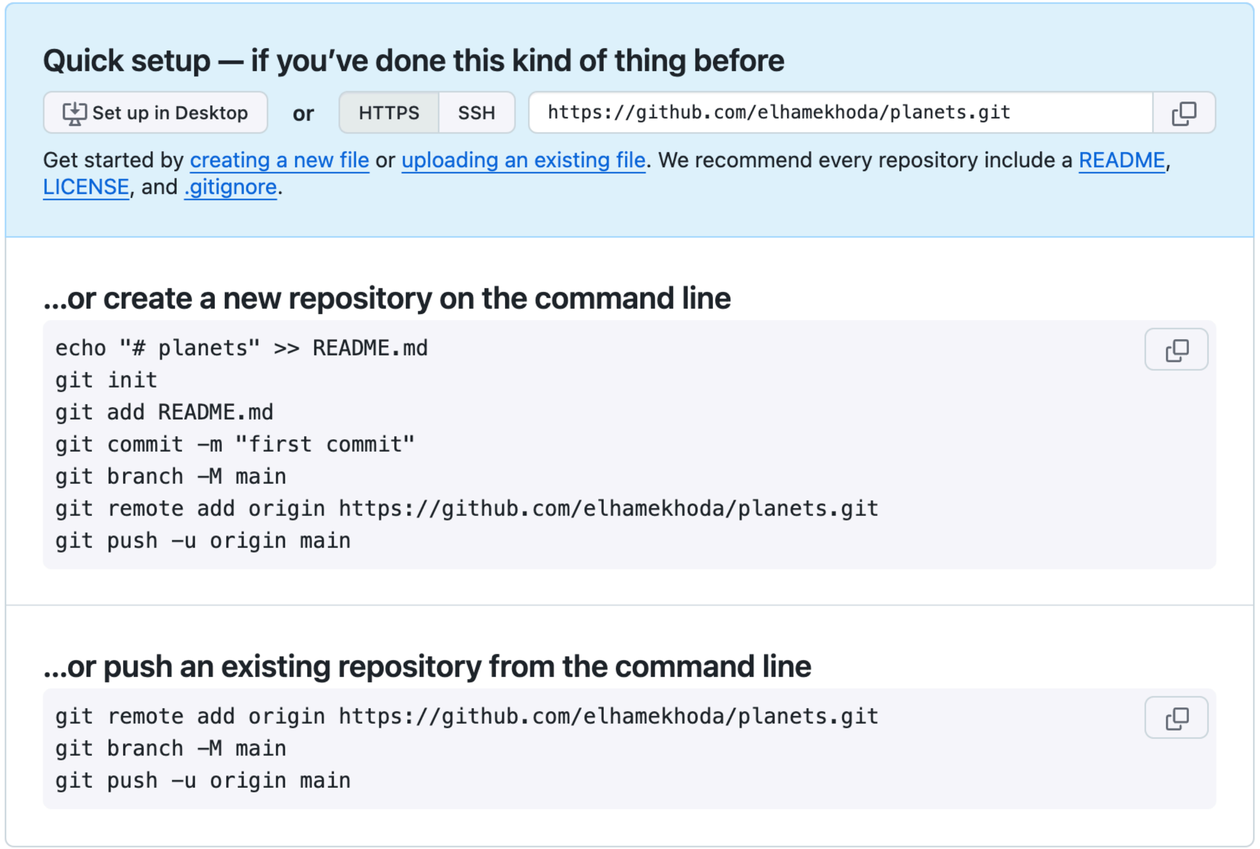


Create a new GitHub repository



How to make your GitHub repository a remote?

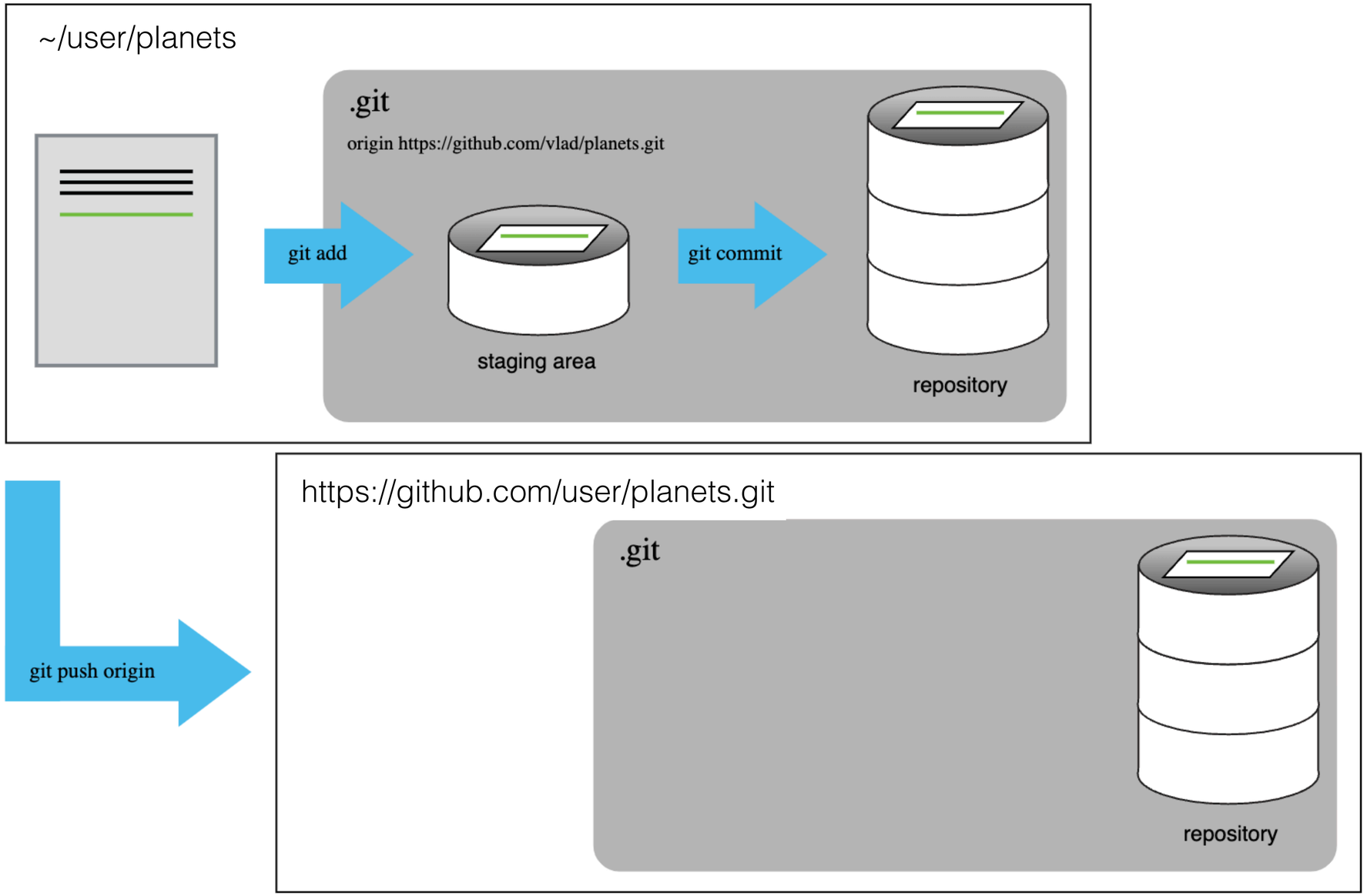




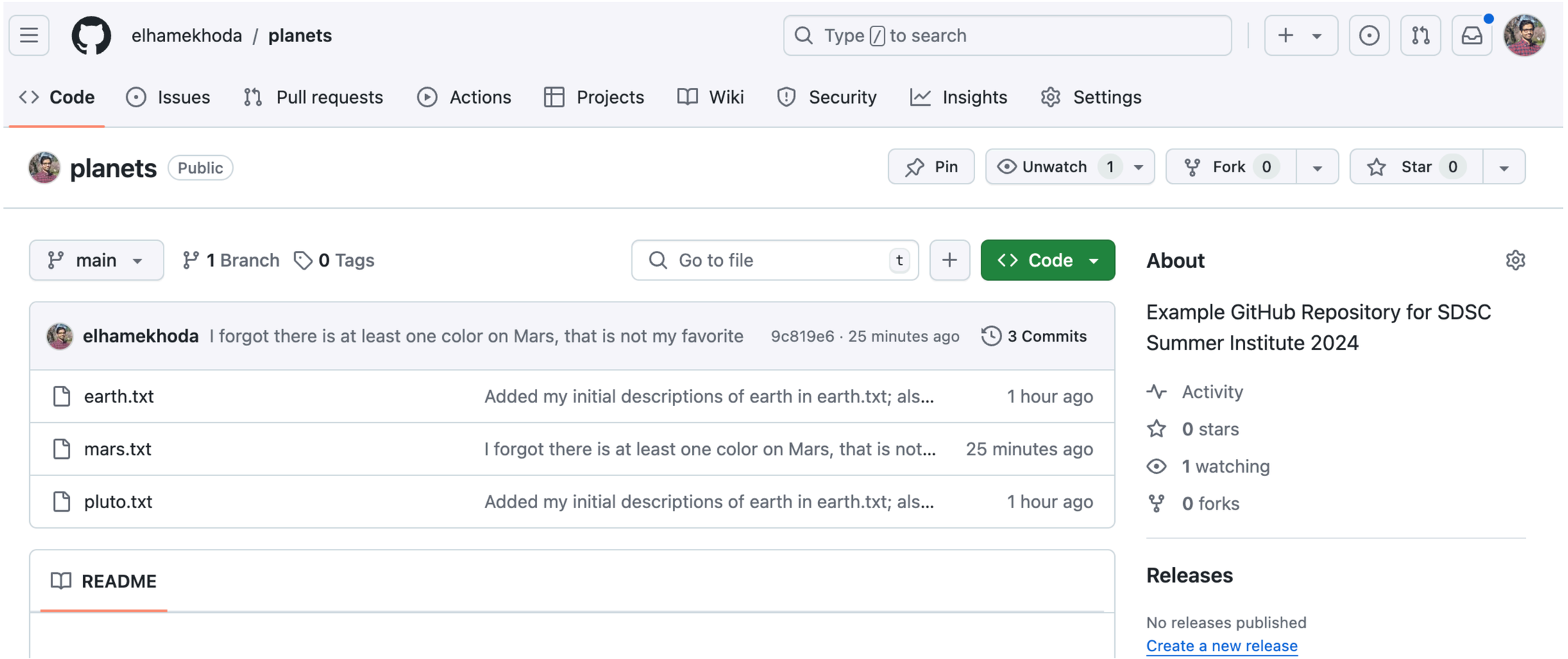
Lets push our changes to remote

```
[elham@login02 planets]$ git remote add origin https://github.com/elhamekhoda/planets.git
[elham@login02 planets]$ git remote -v
origin  https://github.com/elhamekhoda/planets.git (fetch)
origin  https://github.com/elhamekhoda/planets.git (push)
[elham@login02 planets]$ git branch -a
* main
[elham@login02 planets]$ git push -u origin main
Username for 'https://github.com': elhamekhoda
Password for 'https://elhamekhoda@github.com':
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 64 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (10/10), 1.03 KiB | 1.03 MiB/s, done.
Total 10 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/elhamekhoda/planets.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
[elham@login02 planets]$ git branch -a
* main
  remotes/origin/main
[elham@login02 planets]$
```

GitHub repository after the first push



GitHub repository after the first push



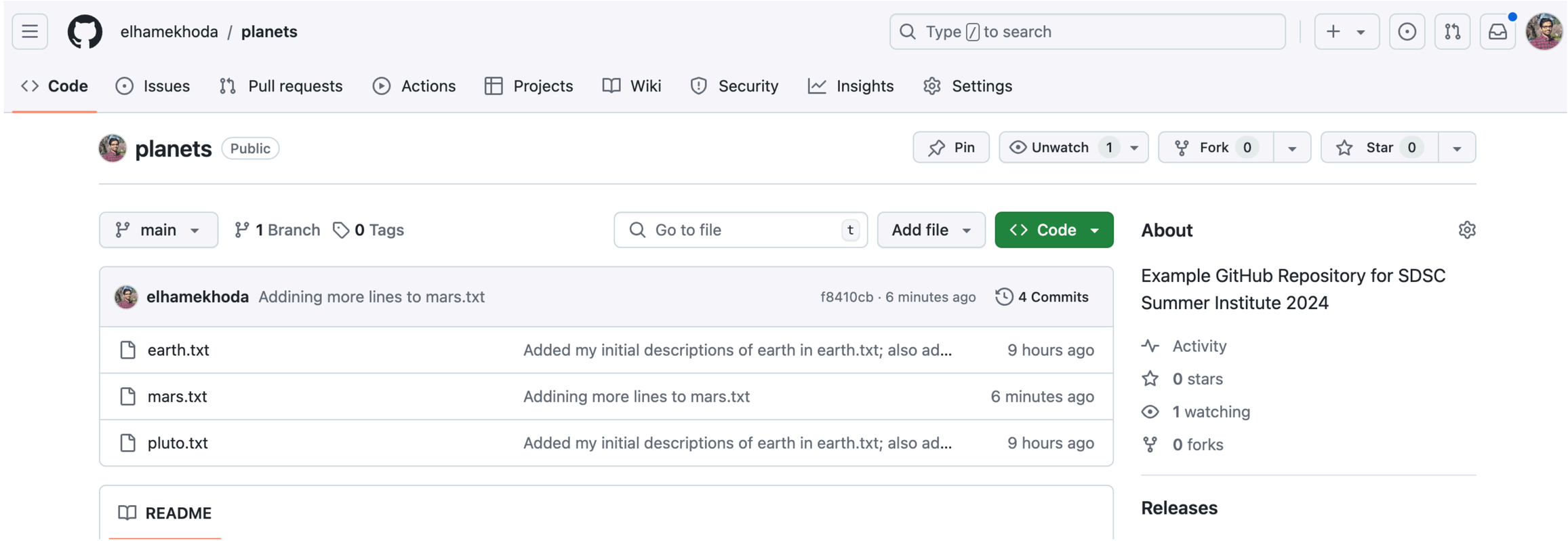
Cloning the GitHub repository with git clone

```
elham@Elhams-MBP git_intro % git clone git@github.com:elhamekhoda/planets.git
Cloning into 'planets'...
Enter passphrase for key '/Users/elham/.ssh/id_rsa':
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 10 (delta 1), reused 10 (delta 1), pack-reused 0
Receiving objects: 100% (10/10), done.
Resolving deltas: 100% (1/1), done.
elham@Elhams-MBP git_intro %
elham@Elhams-MBP git_intro %
elham@Elhams-MBP git_intro % ls
intro_git_github_slides.key     planets
elham@Elhams-MBP git_intro % cd planets
elham@Elhams-MBP planets % ls
earth.txt       mars.txt        pluto.txt
elham@Elhams-MBP planets % git config --list
credential.helper=osxkeychain
init.defaultbranch=main
user.name=Elham E Khoda
user.email=elhamekhoda@gmail.com
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
core.ignorecase=true
core.precomposeunicode=true
remote.origin.url=git@github.com:elhamekhoda/planets.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.main.remote=origin
branch.main.merge=refs/heads/main
elham@Elhams-MBP planets %
elham@Elhams-MBP planets % git log -1
commit 9c819e63761cccea804ff7dbdfab2d2e0af3dac5 (HEAD -> main, origin/main, origin/HEAD)
Author: Elham E Khoda <elhamekhoda@gmail.com>
Date:   Tue Aug 6 22:57:03 2024 -0700

    I forgot there is at least one color on Mars, that is not my favorite
elham@Elhams-MBP planets %
```

Modify and push changes back to GitHub

```
elham@Elhams-MBP planets %
elham@Elhams-MBP planets % git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
elham@Elhams-MBP planets %
elham@Elhams-MBP planets % echo "It's where human wants to move" >> mars.txt
elham@Elhams-MBP planets % git status
On branch main
Your branch is up to date with 'origin/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:   mars.txt

no changes added to commit (use "git add" and/or "git commit -a")
elham@Elhams-MBP planets % git add mars.txt
elham@Elhams-MBP planets % git commit -m "Addining more lines to mars.txt"
[main f8410cb] Addining more lines to mars.txt
 1 file changed, 1 insertion(+)
elham@Elhams-MBP planets % git push
Enter passphrase for key '/Users/elham/.ssh/id_rsa':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 10 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 378 bytes | 378.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:elhamekhoda/planets.git
   9c819e6..f8410cb  main -> main
elham@Elhams-MBP planets %
```

GitHub repository after the push



Create and Switch between branches

When you want to begin modifying the contents of your repository, but only want to do so to a limited subset of the files in it, you can begin using the branching and merging mechanisms of git.

To create a new branch in your repository, you use the command:

```
git branch <branch-name>
```

To then begin working on this new branch, you must switch over to this branch of this project using the command:

```
git checkout <branch-name>
```

The default branch in every git repository is the master (or main) branch.

Create and Switch between branches

```
nothing to commit, working tree clean
[elham@login01 planets]$ git branch
* main
[elham@login01 planets]$ git branch earth
[elham@login01 planets]$ git checkout earth
Switched to branch 'earth'
[elham@login01 planets]$ git branch
* earth
  main
[elham@login01 planets]$ echo "It is not flat." >> earth.txt
[elham@login01 planets]$ git status
On branch earth
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:   earth.txt

no changes added to commit (use "git add" and/or "git commit -a")
[elham@login01 planets]$ git add earth.txt
[elham@login01 planets]$ git commit -m "creating new branch for earth.txt and adding more info to that file"
[earth 5f8989c] creating new branch for earth.txt and adding more info to that file
 1 file changed, 1 insertion(+)
[elham@login01 planets]$ git log
commit 5f8989c1775a938daef45f70c60ee652543cc697 (HEAD -> earth)
Author: Elham E Khoda <elhamekhoda@gmail.com>
Date:   Wed Aug 7 07:26:41 2024 -0700

    creating new branch for earth.txt and adding more info to that file

commit 9c819e63761cccea804ff7dbdfab2d2e0af3dac5 (origin/main, main)
Author: Elham E Khoda <elhamekhoda@gmail.com>
Date:   Tue Aug 6 22:57:03 2024 -0700

    I forgot there is at least one color on Mars, that is not my favorite

commit fe45fcff646078a1ea9f8fcb72dc5223952348bd
Author: Elham E Khoda <elhamekhoda@gmail.com>
Date:   Tue Aug 6 21:53:55 2024 -0700

    Added my initial descriptions of earth in earth.txt; also added my initial descriptions of pluto in pluto.txt,
```

Summary

Archiving: You must regularly save the changes you make.

Reproducibility: Creating a history of saved changes allows you to revert selected files or your entire project back to any previous state in its recorded history.

Collaboration: You and a team of contributors can work on a project independently, but share your changes amongst one another as the project takes shape.

Accountability: Compare changes, see who last modified something, and who introduced a problem and when.

Recoverability: Each contributor has their own local, recent copy of the project and its complete history, making it highly unlikely you'll ever lose a significant portion of the project

References

Version Control with Git
by Erin Graham, Katherine Koziar, Martino Sorbaro
https://swcarpentry.github.io/git-novice/

Pragmatic Version Control Using Git
by T. Swicegood

Pro Git by S. Chacon and B. Straub
https://git-scm.com/book/en/v2