Version Control with **`git`**

Advanced Programming Techniques
Fall 2013

What is version control?

- Track the history of a collection of files—most often source code.
- Allows us to:
  - See what files changed and when they changed.
  - Compare (diff) two or more versions.
  - Recover (check out) old version.
  - Experiment with new ideas without the risk of losing existing work (branching).
- Greatly facilitates collaboration.

Why version control?

- Have you ever taken a working program and introduced a bug?
- Have you ever taken a working program and introduced a bug *but not noticed immediately*?

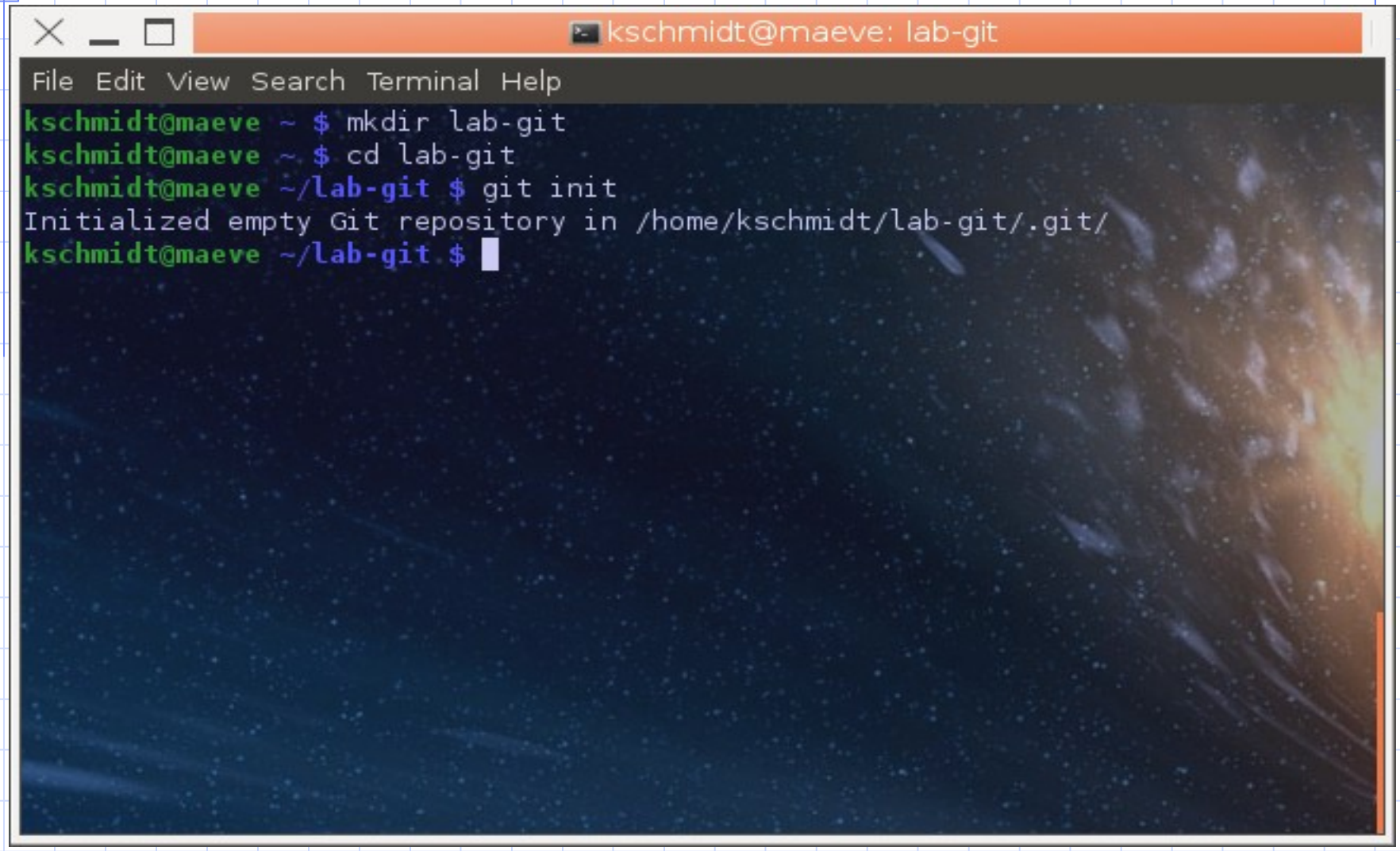
Version control with **`git`**

- Many version control systems: bazaar, cvs, darcs, mercurial, Perforce, subversion, Visual SourceSafe, etc.
- But **`git`** has (largely) won.
  - Developed to manage Linux kernel source code.
  - Popularized by github.
  - Widely used

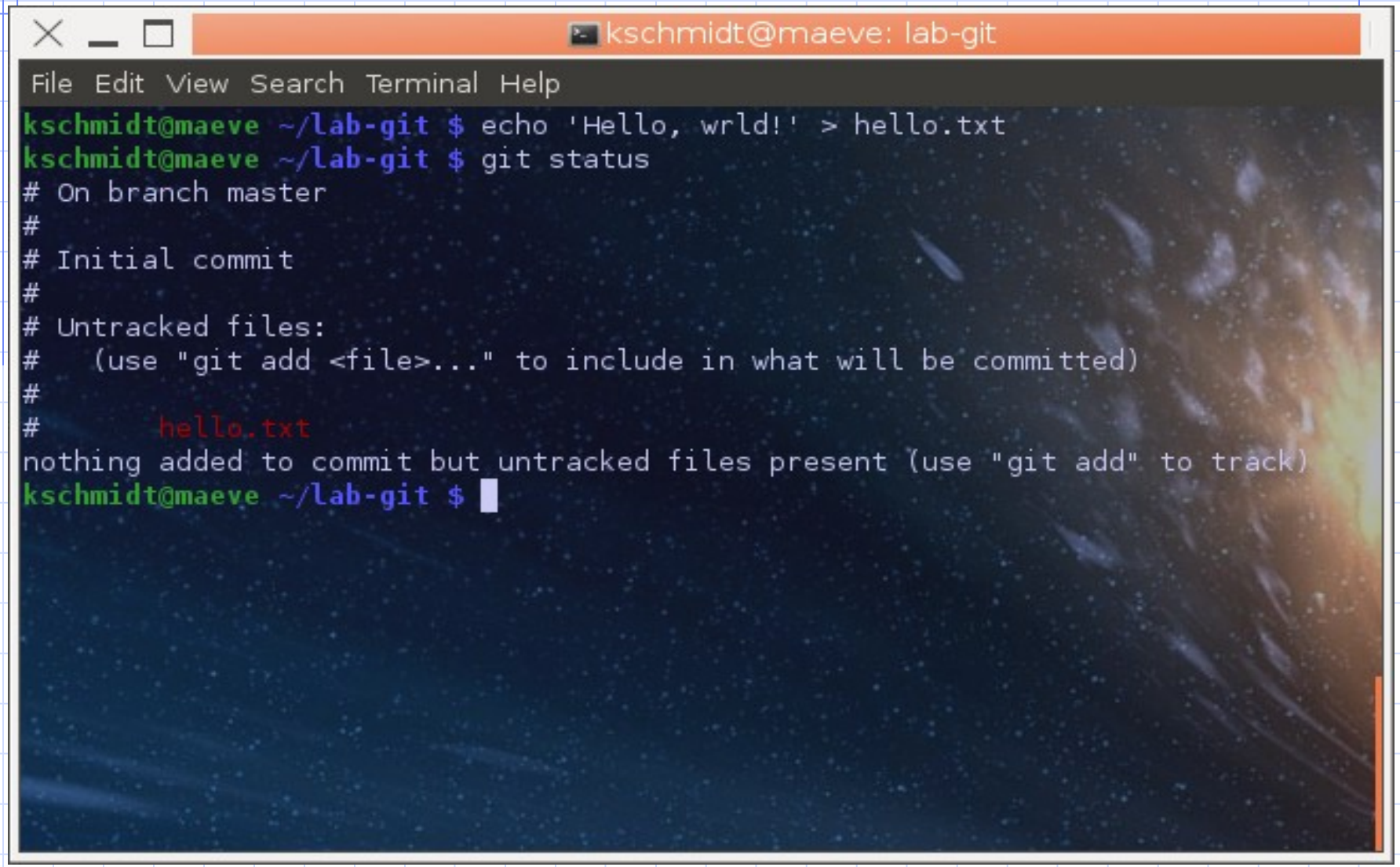
`git` mini-lab – HOLD (TODO)

- For *this portion of the lab only*, you will submit your work by checking it into a **`git`** repository that lives on **`tux`**.

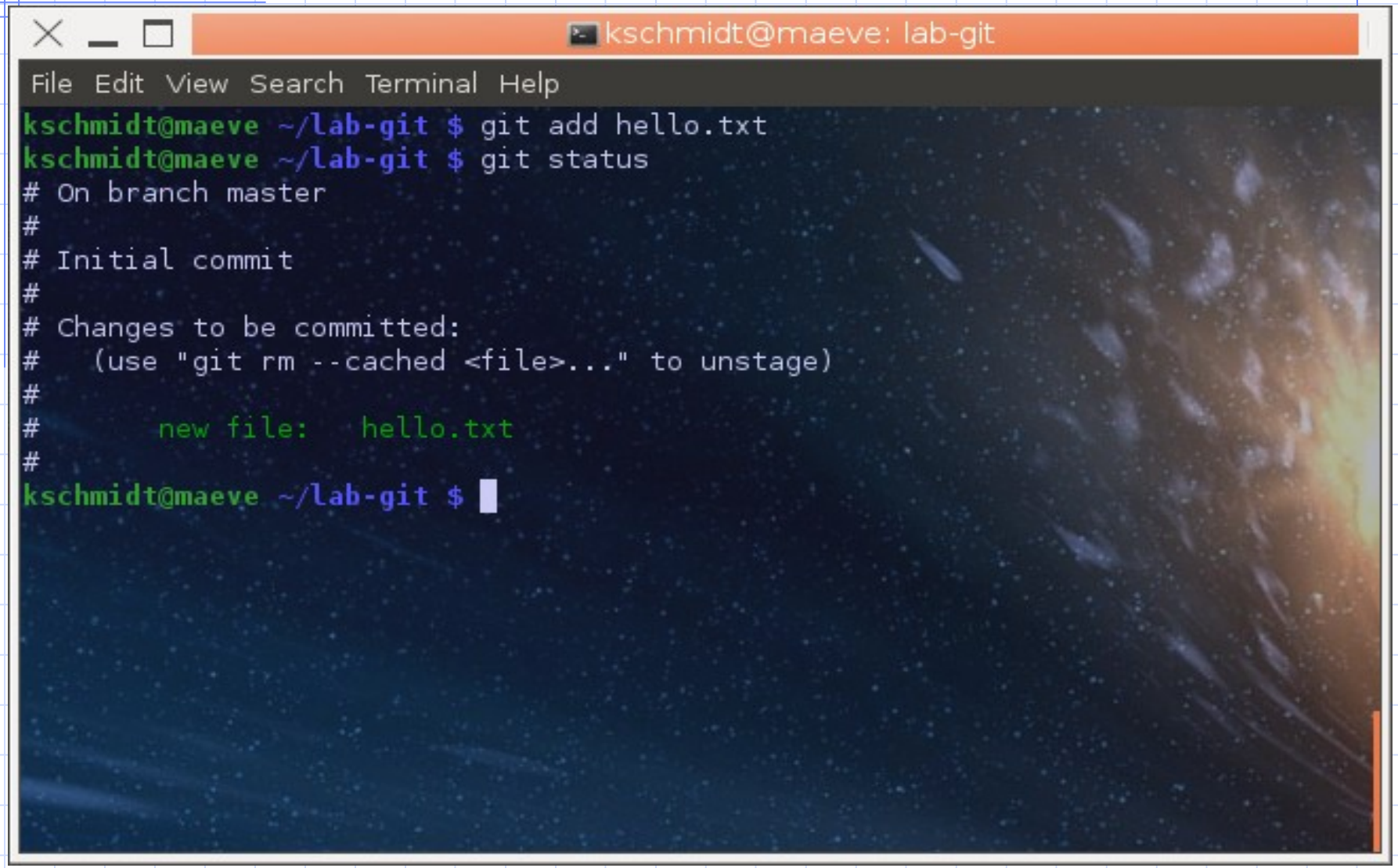
Creating a Repository – `git init`



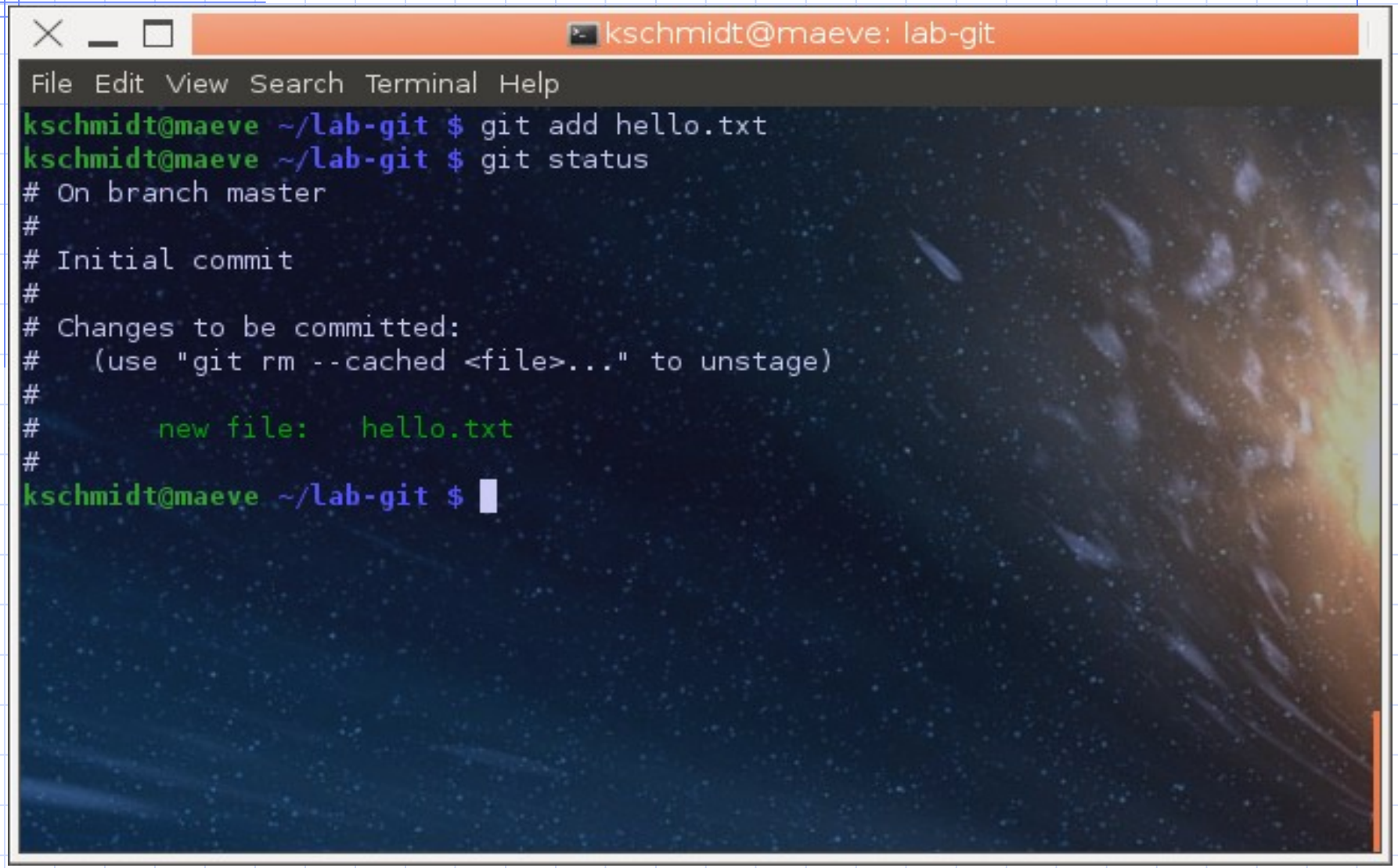
Adding a file – git add



Adding a file (cont'd)



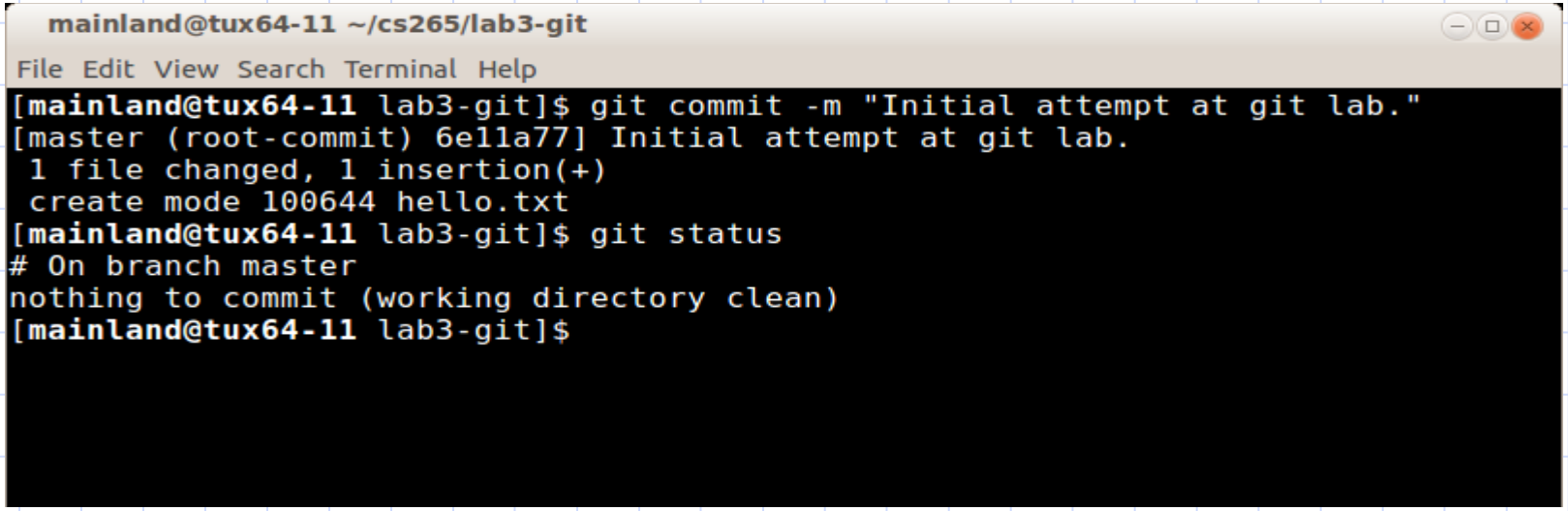
Adding a file (cont'd)



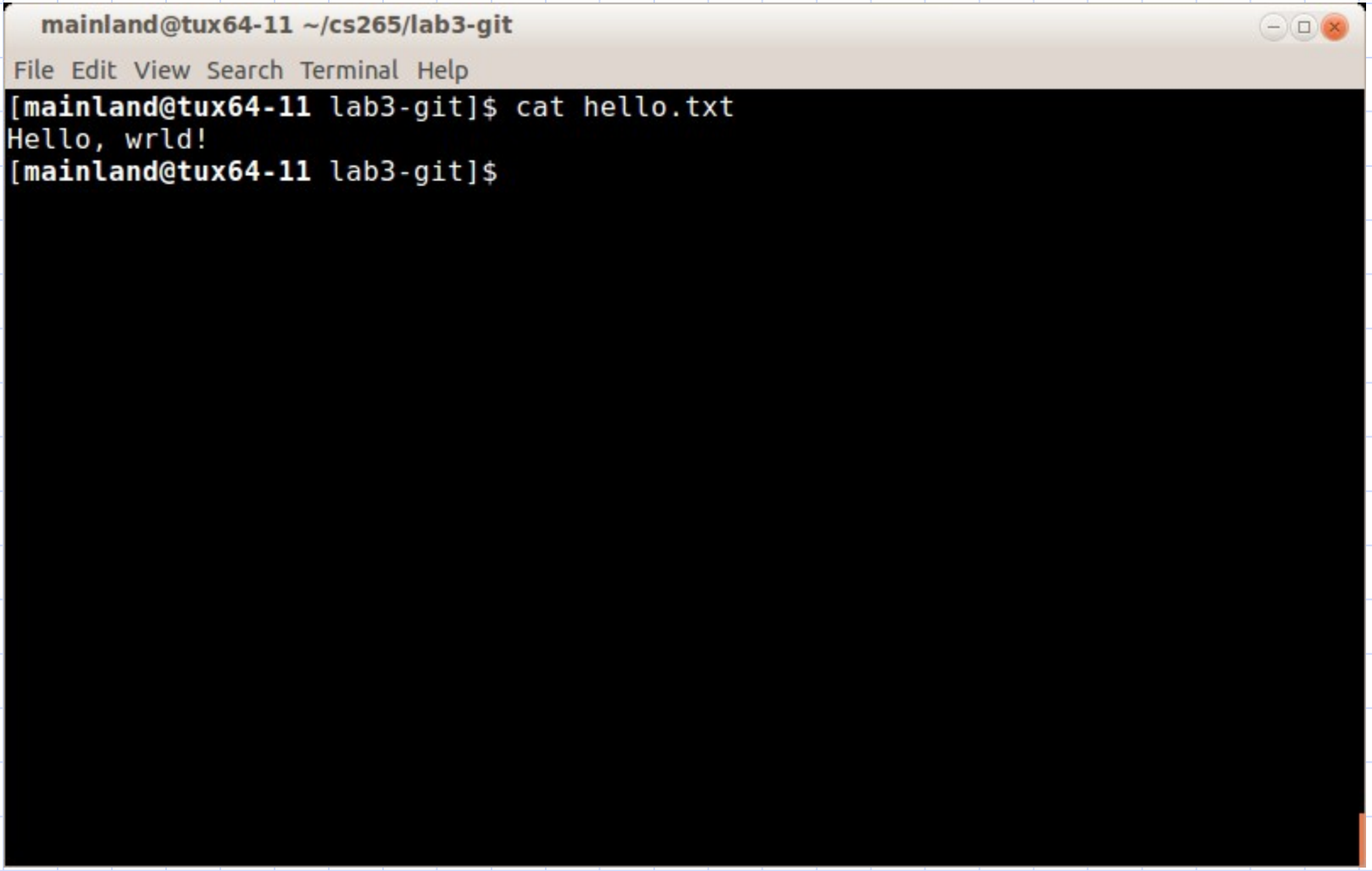
Committing our work

`git commit [-m <msg>]`

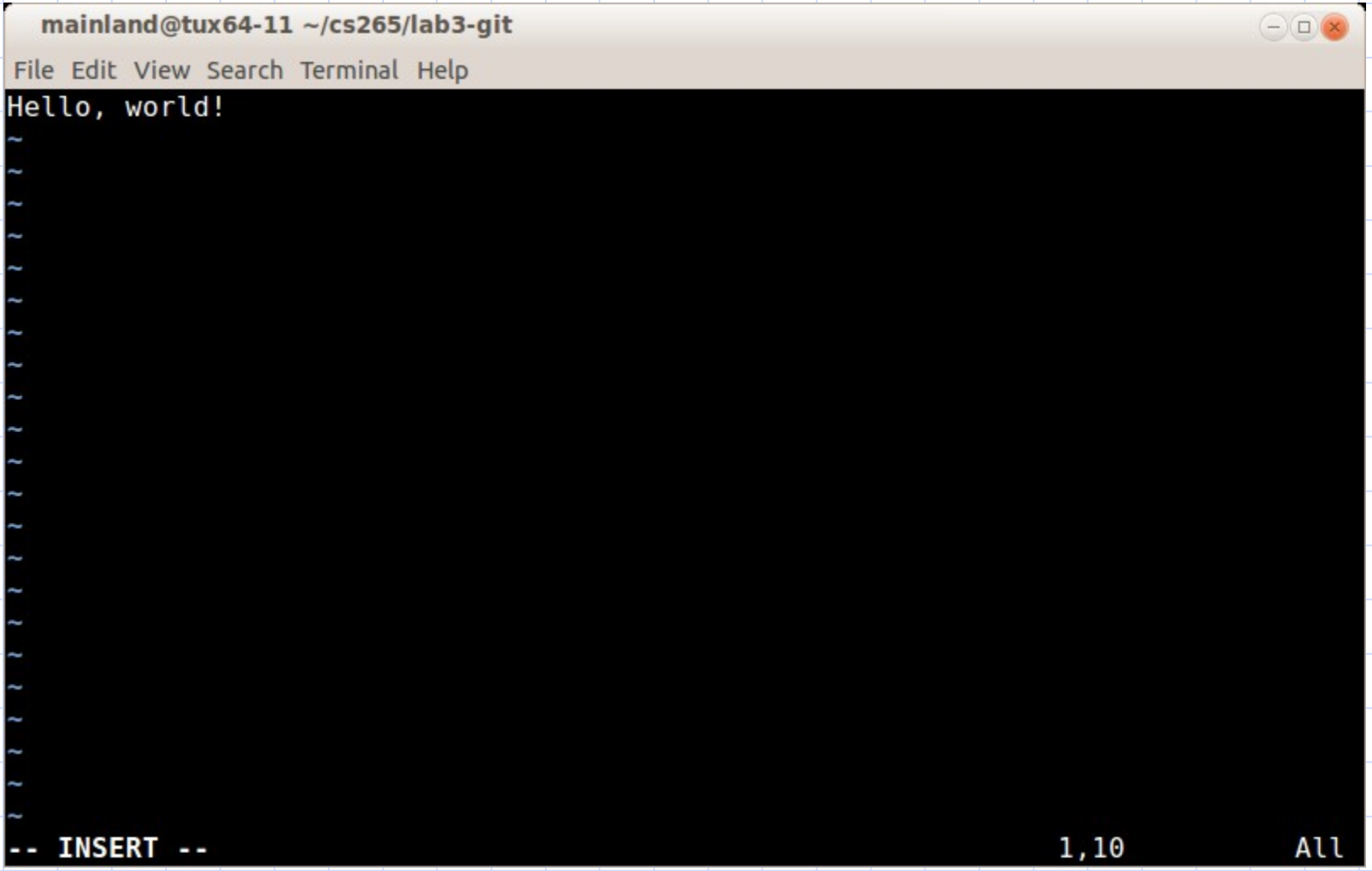
- Commits changes to the working directory into the repository
- If a msg is not supplied, the log will be opened in an editor



Oops...


mainland@tux64-11 ~/cs265/lab3-git
File Edit View Search Terminal Help
[mainland@tux64-11 lab3-git]$ cat hello.txt
Hello, wrld!
[mainland@tux64-11 lab3-git]$

Making Changes



That's better...

- Changes to current working directory can be compared to the repository

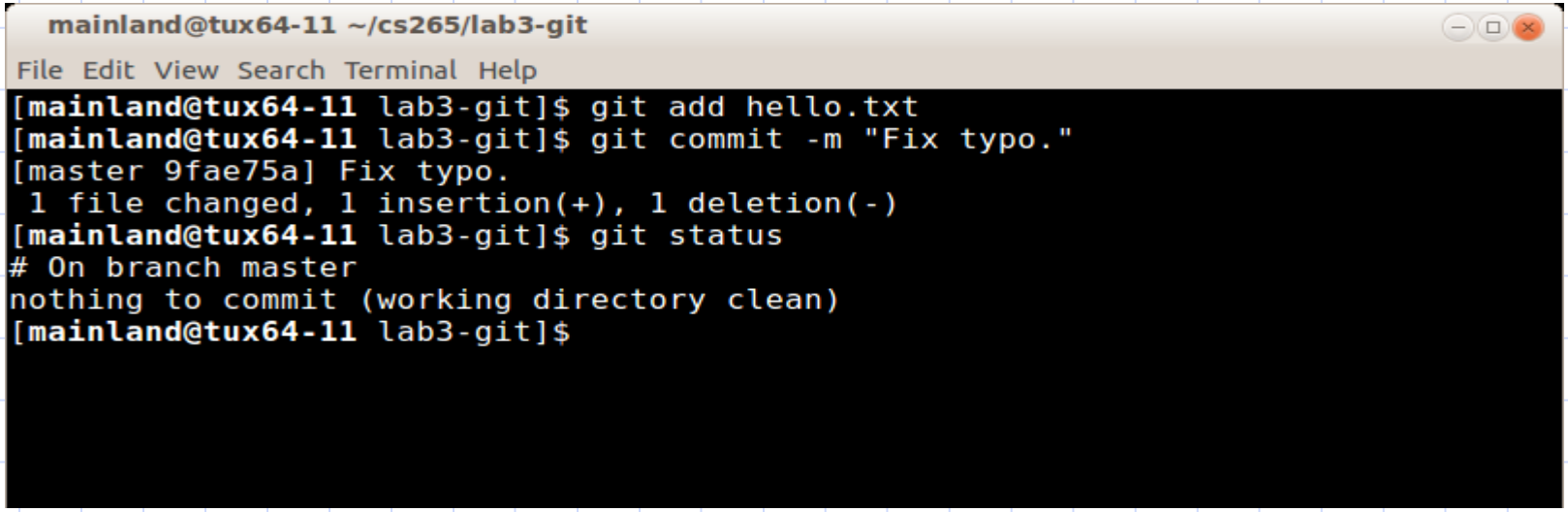
```
git diff
```

```
mainland@tux64-11 ~/cs265/lab3-git
File  Edit  View  Search  Terminal  Help
[mainland@tux64-11 lab3-git]$ 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:   hello.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
[mainland@tux64-11 lab3-git]$ git diff
diff --git a/hello.txt b/hello.txt
index 26899e5..af5626b 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1 +1 @@
-Hello, wrld!
+Hello, world!
[mainland@tux64-11 lab3-git]$
```

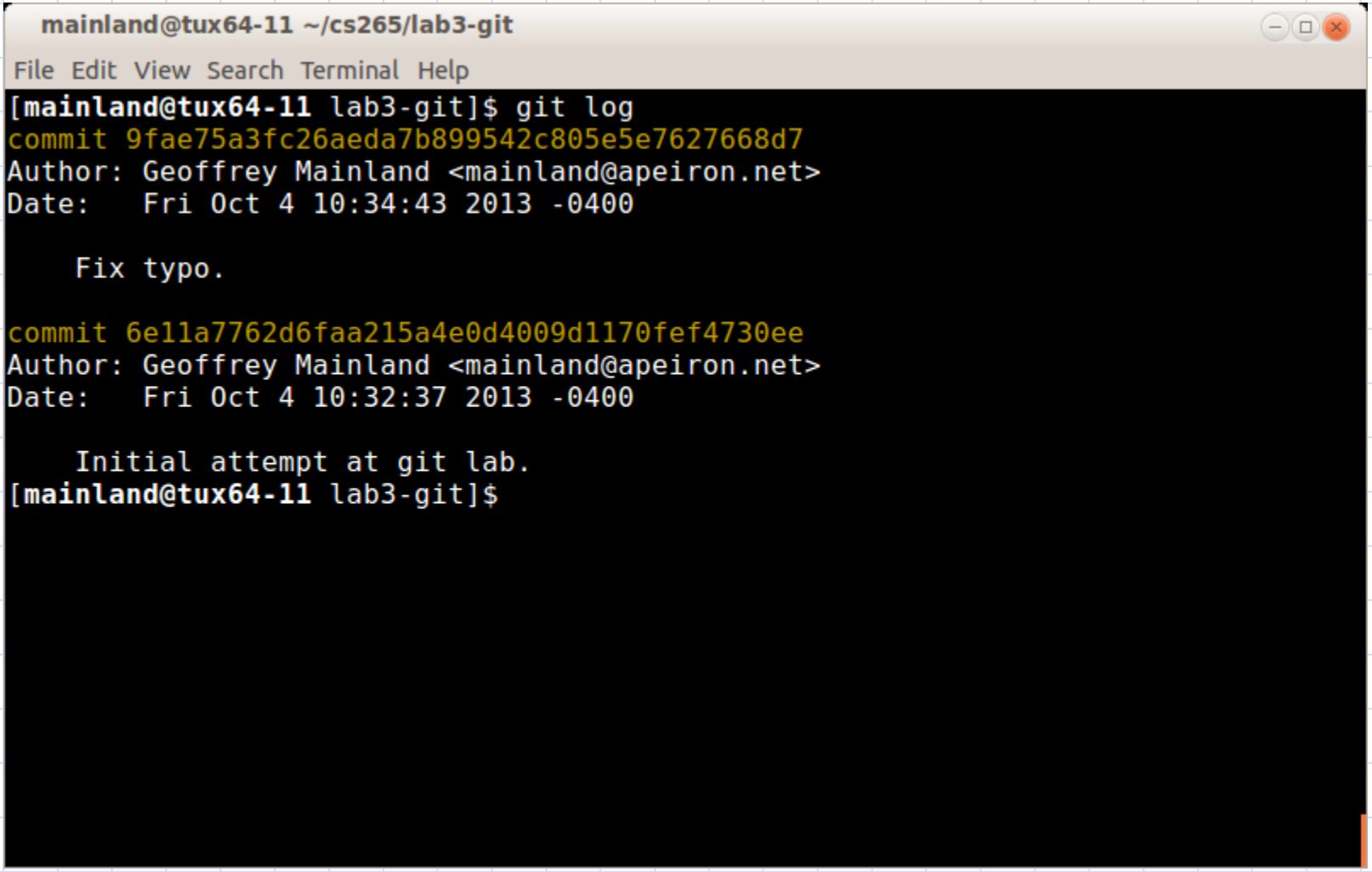
Committing changes

- Changes to files must be added to the index
- Changes to known files can also be added with the update flag
  `git add -u`

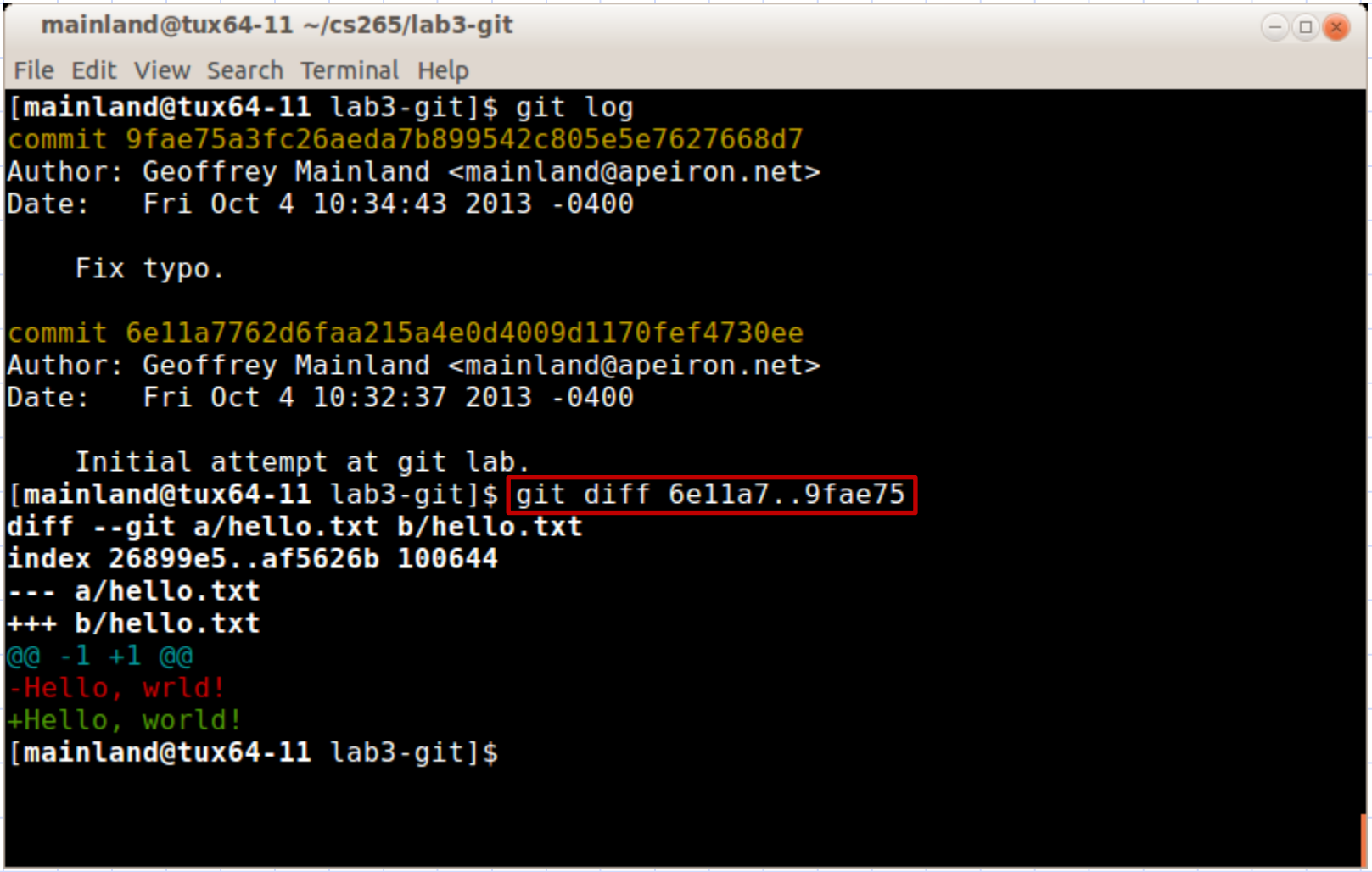
- Then commit changes



Examining what we've done.



Examining what we've done.



Summary

Initialize a new repo	`git init`
Check repo status	`git status`
Add a file	`git add <file>`
Commit changes	`git commit` `git commit -m <msg>`
See commit history	`git log`
Diff versions	`git diff` `git diff A..B`

Getting help in **git**

```
git help <topic>
```

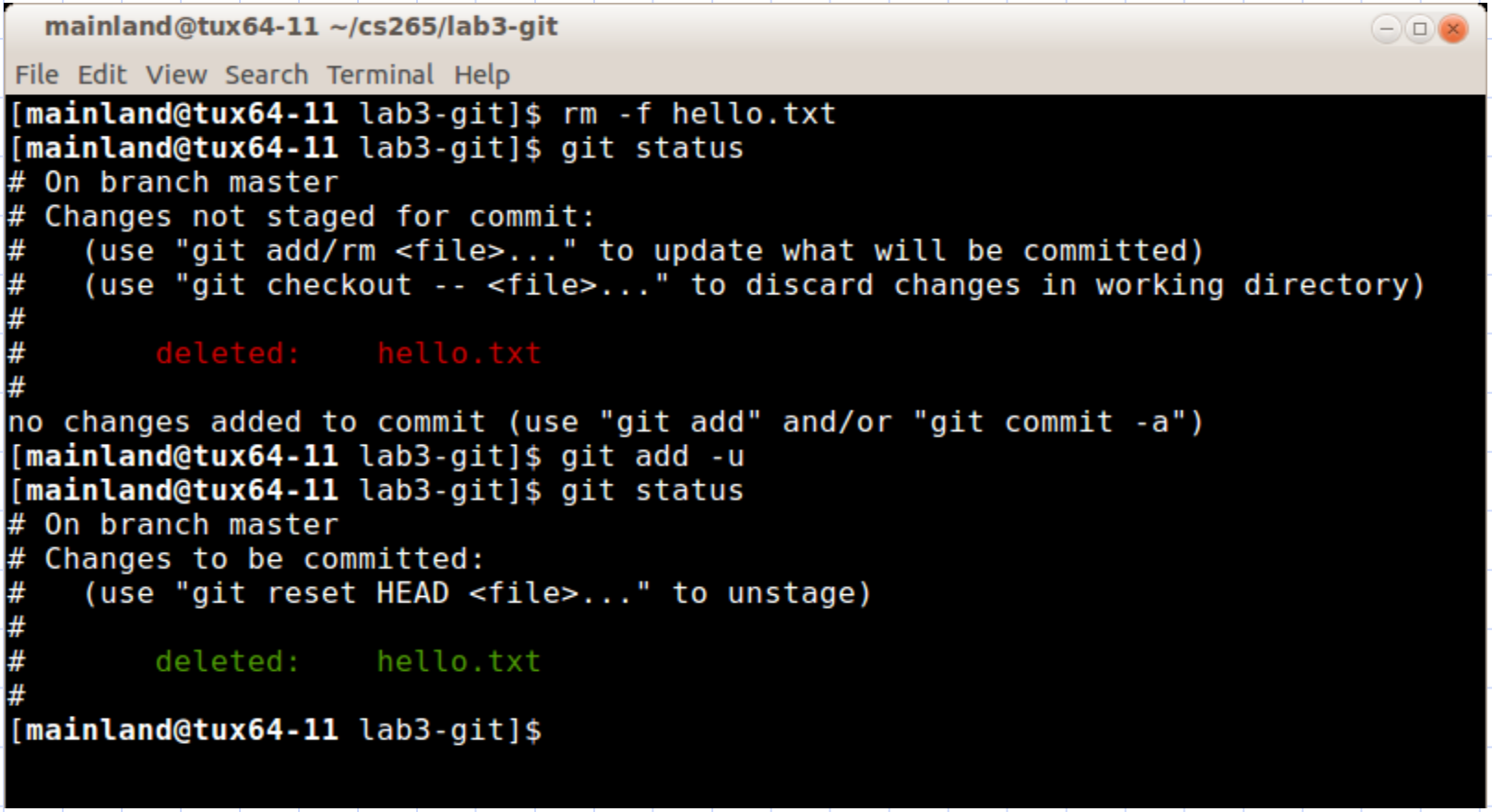
- git will display help for this topic

Some `git` concepts

- The **working directory** is the directory where you've checked out code.
- The **index** is the set of changes that are **staged** and will become the next commit.
  - The index may be different from the working directory!

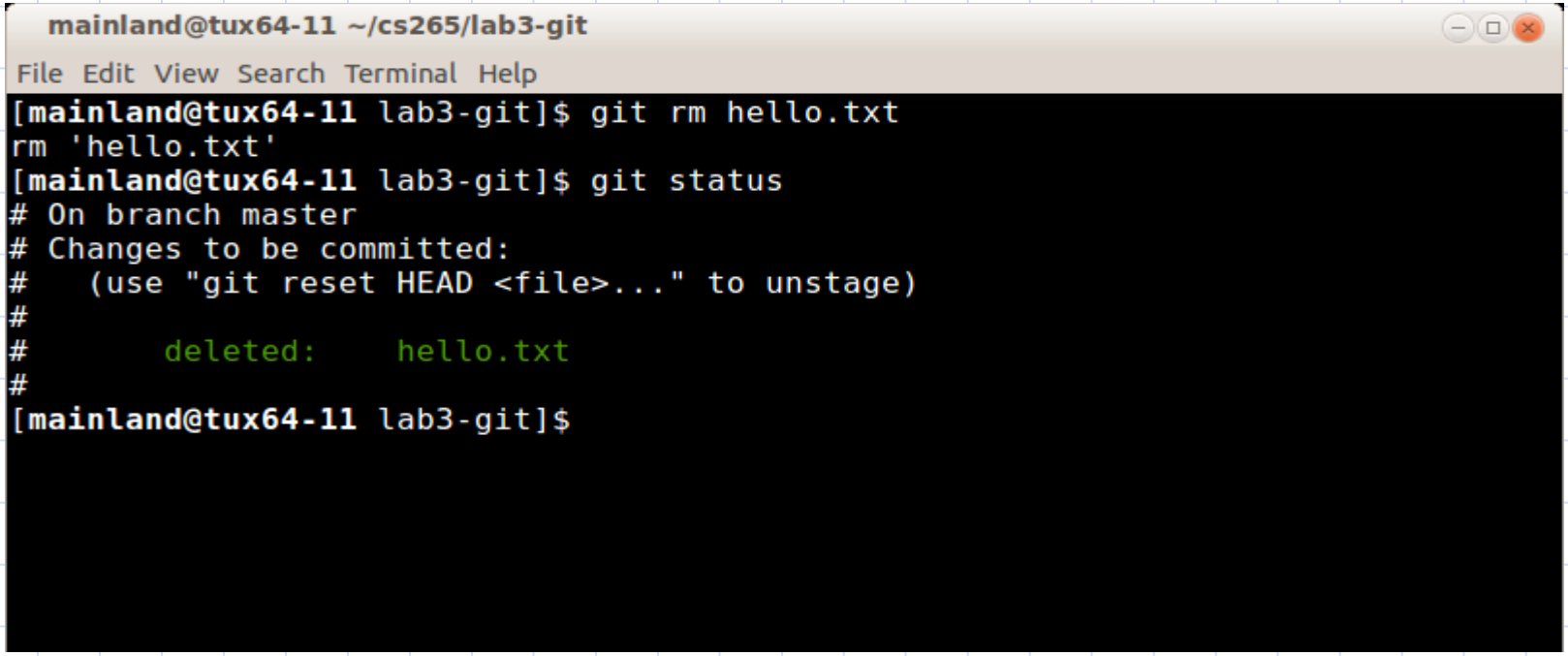
Removing a file

- Just delete the file, add the change to the index, then commit, OR....



Removing a file, take 2

- Use git to remove the file
  `git rm <file>`
- (Then commit the changes to the repository



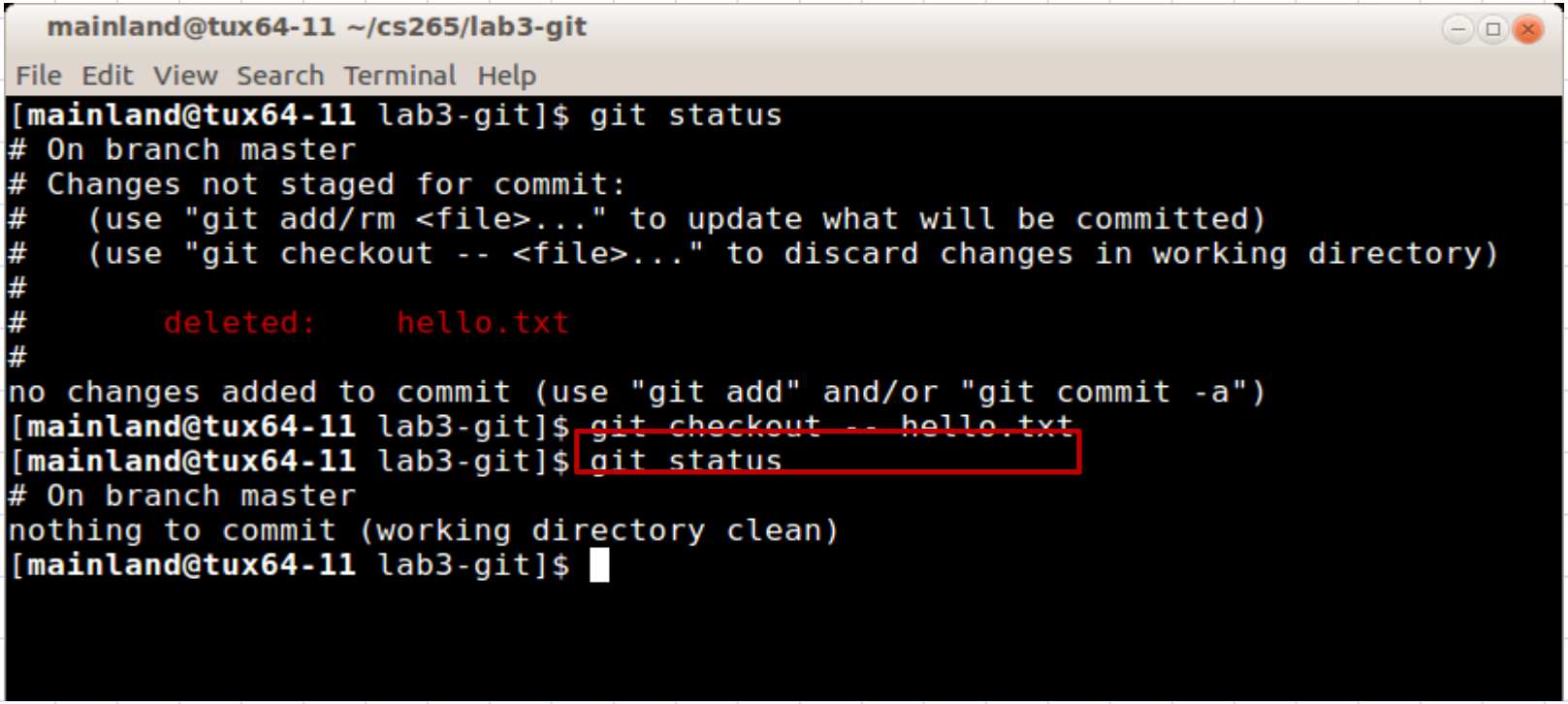
Undoing mistakes

- git reset removes all pending changes (clears the index)
  - Does *not* touch the files in the working directory

```
File  Edit  View  Search  Terminal  Help
[mainland@tux64-11 lab3-git]$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       deleted:    hello.txt
#
[mainland@tux64-11 lab3-git]$ git reset
Unstaged changes after reset:
D       hello.txt
[mainland@tux64-11 lab3-git]$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add/rm <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       deleted:    hello.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
[mainland@tux64-11 lab3-git]$
```

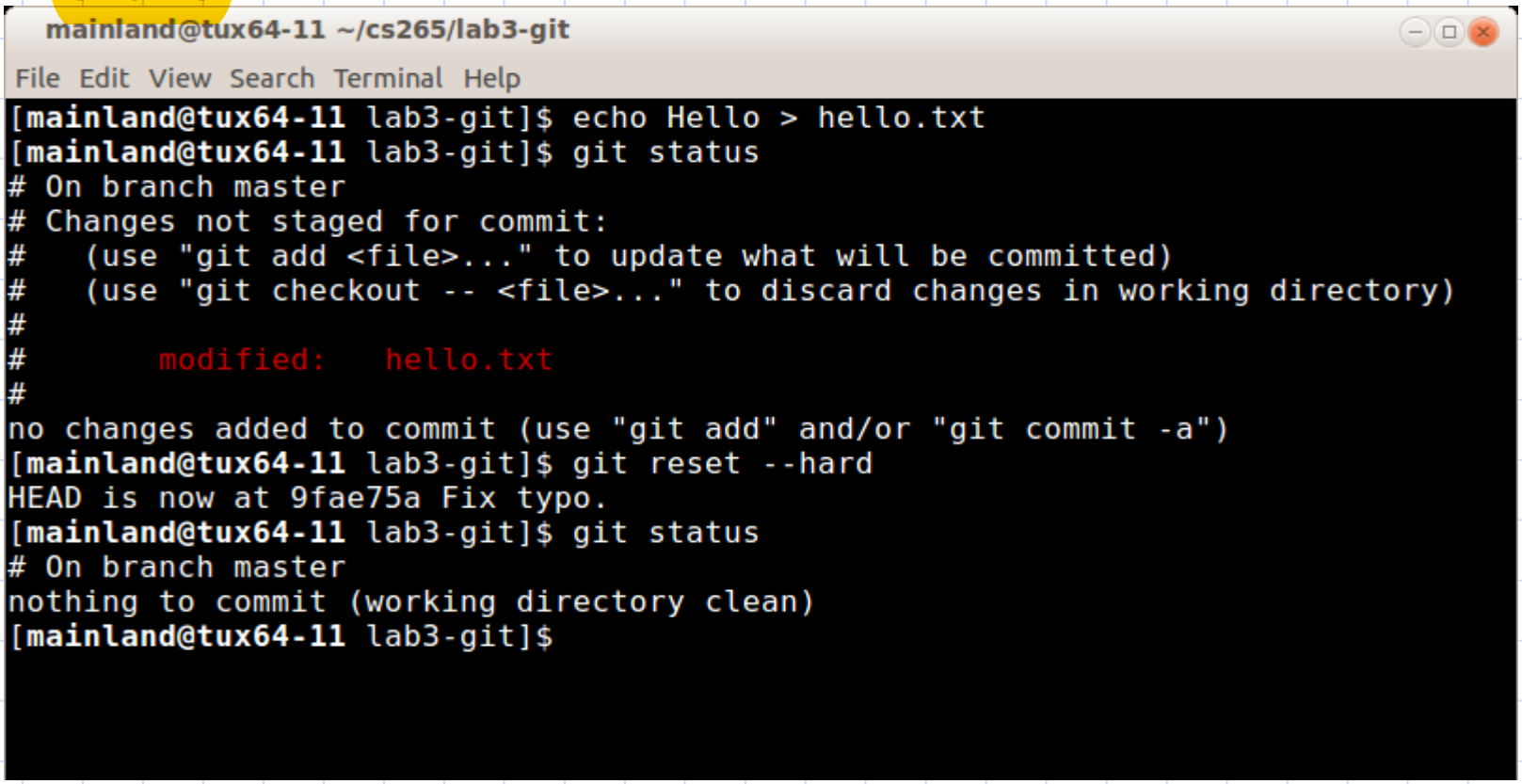
Undoing mistakes, cont'd

- `git checkout` can be used to pull the last (committed) version from the repository to the working directory



Reverting all changes

- `git reset --hard` puts the repository and working directory back to the last committed state



Summary

Remove a file	`git rm <FILE>`
Add changes to index	`git add -u`
Reset index	`git reset`
Discard changes to working directrory	`git checkout -- <FILE>`
Discard all changes	`git reset --hard`

`git` is distributed

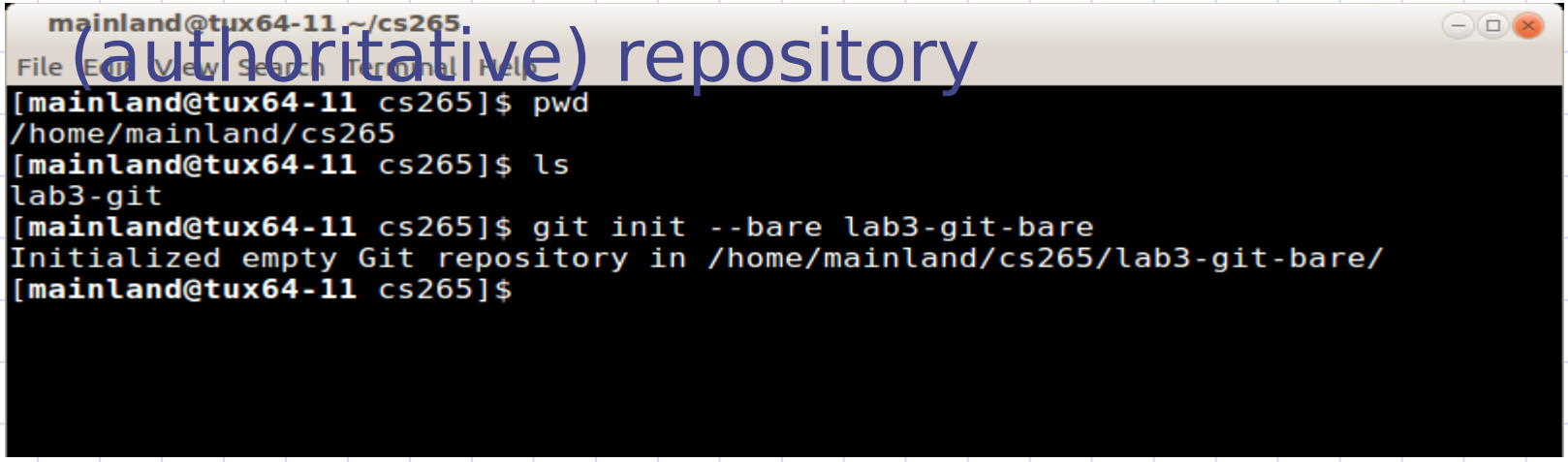
- Peer-to-peer vs. client-server.
- There is no “central” repository (except by convention).
- Each developer's copy of the code is a real repository.
- Synchronize by exchanging *patches* (sets of changes) with other developers.

Repositories and Branches

- A repository contains one or more *branches*
- `git branch [-a|-r]`
- Separate branches are useful
  - Maintaining several releases of a product
  - Experiment, add features, take it for a spin before merging new feature into release
- **Not** this week. Nor this term. Plenty of tutorials out there.

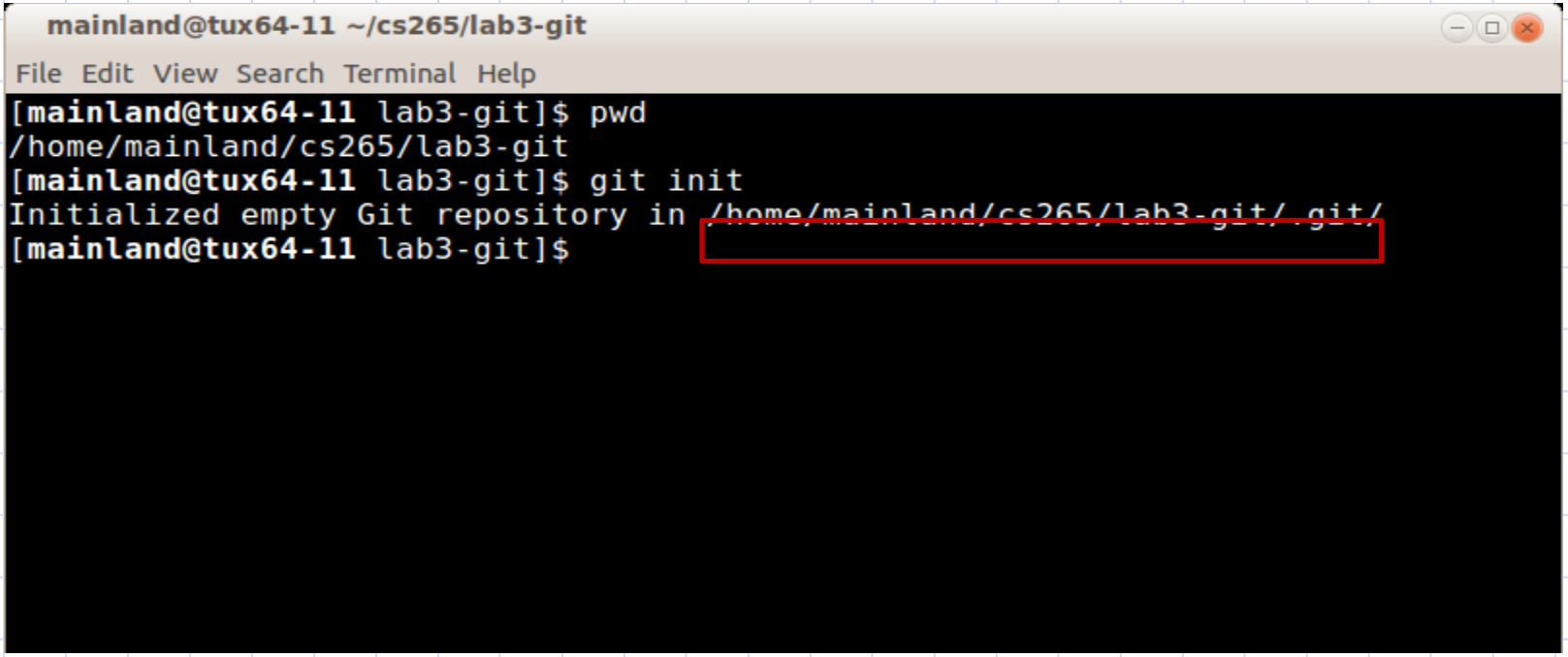
Creating a **bare** repository

- Bare means no working directory.
- Useful for coordinating the movement of patches between other repositories.
- In this example, this will be considered to be the main (authoritative) repository



Let's compare...

- Remember when we created lab3-git

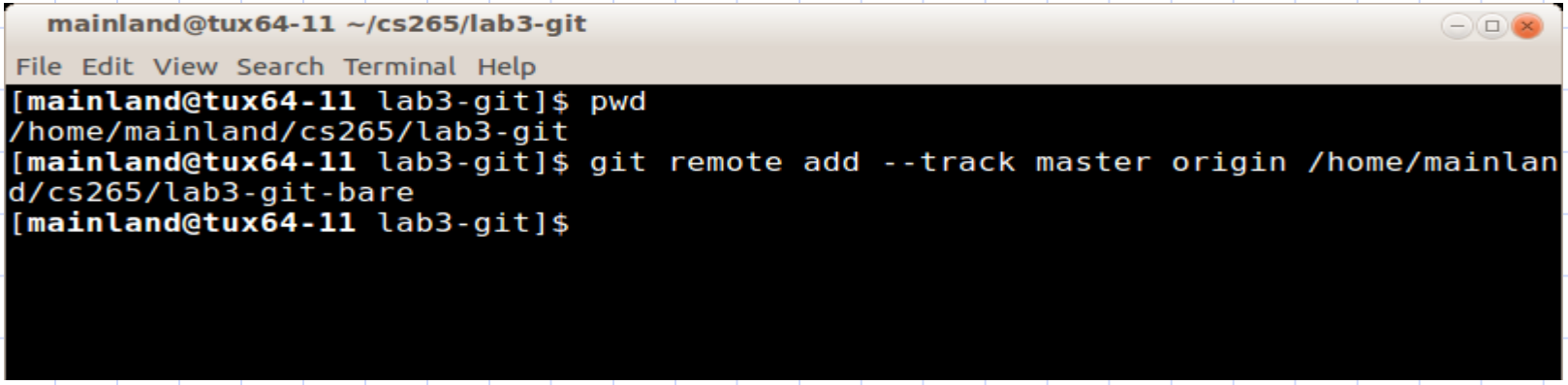


Adding a **remote**

- Distant repository to synch to
  - Others can pull from this, add work

`git remote add --track <branch> <label> <location>`

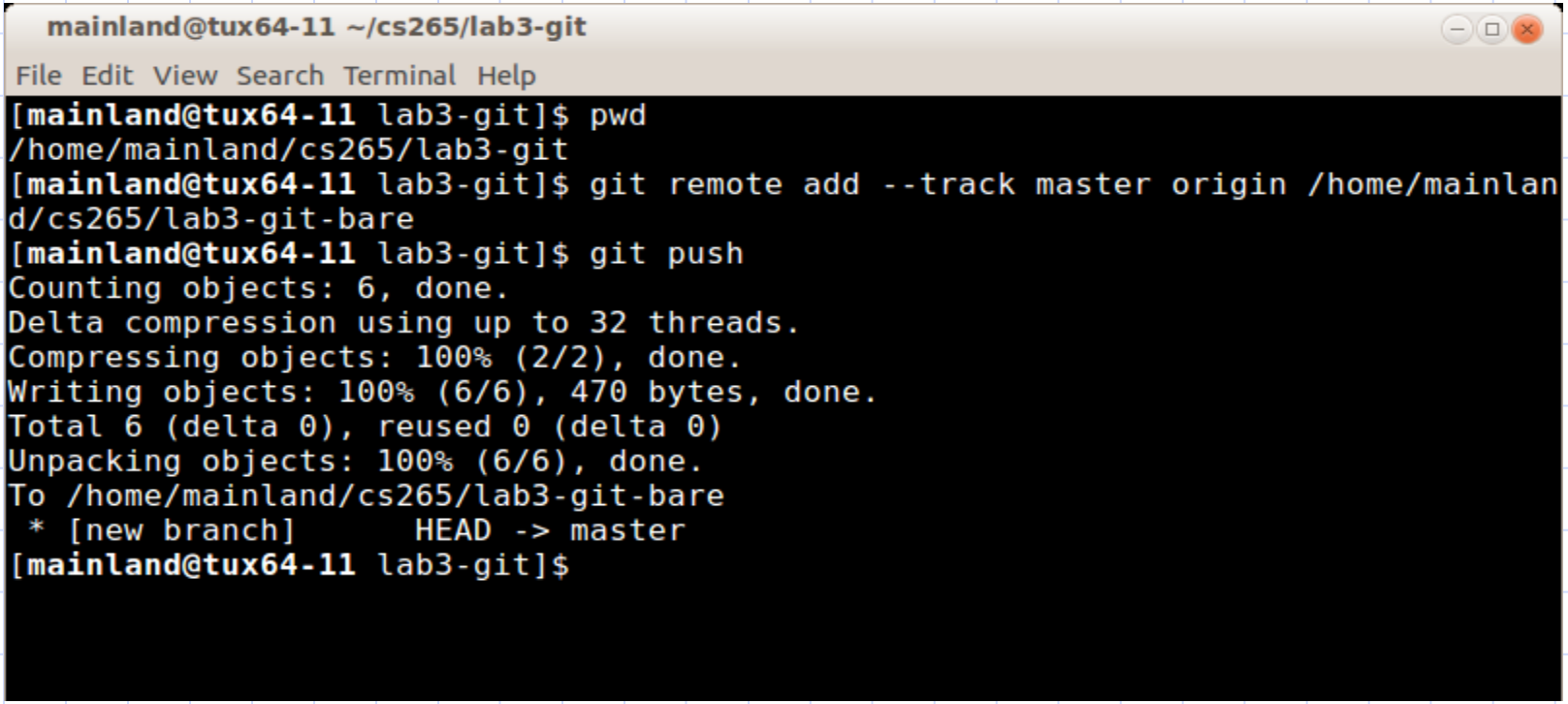
- `<branch>` is the distant branch to be tied to the current local one (`master`)
- `<label>` is the local name (alias) of the distant repository
- `<location>` is the path, URL, etc.



Pushing to a remote

`git push origin master`

- To make the remote branch the upstream branch for local:
  `git branch --set-upstream origin/master`

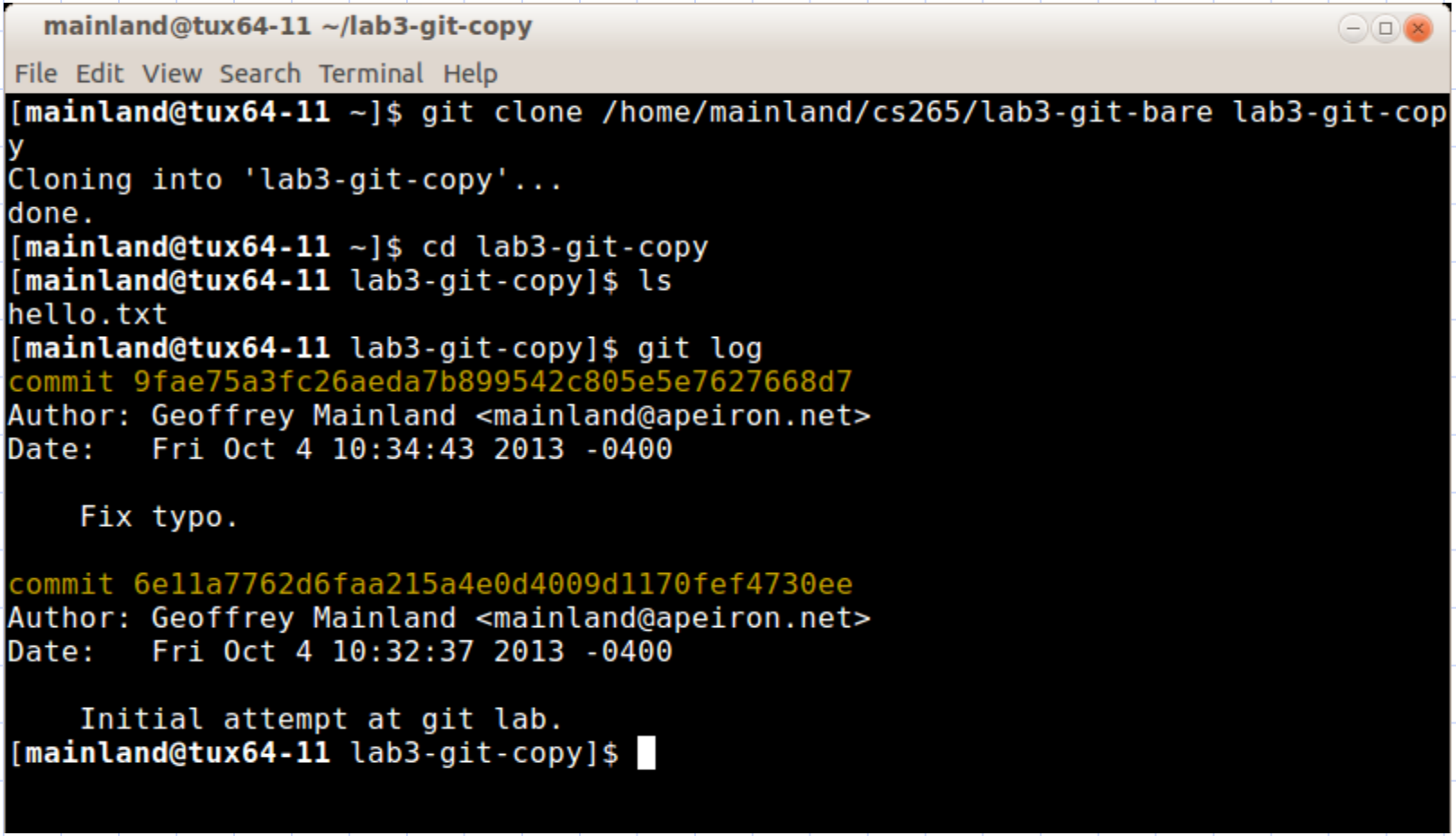


`git` branches

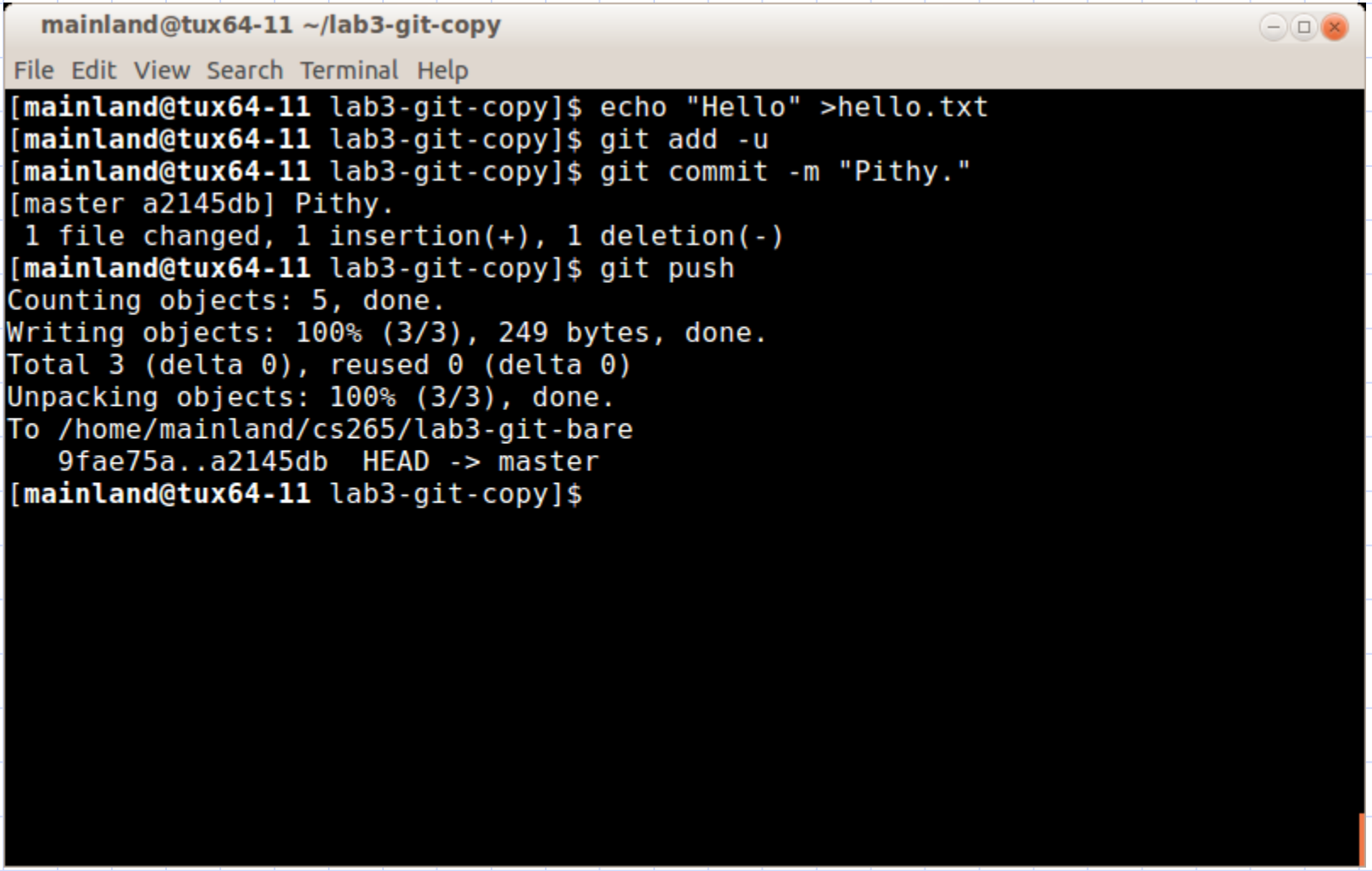
- Branches allow multiple lines of development to occur in the same repository.
- The default branch is named **`master`**.
- The **`--track master`** argument to **`git remote add`** makes **`master`** the default remote branch for pushing.

Cloning a repository

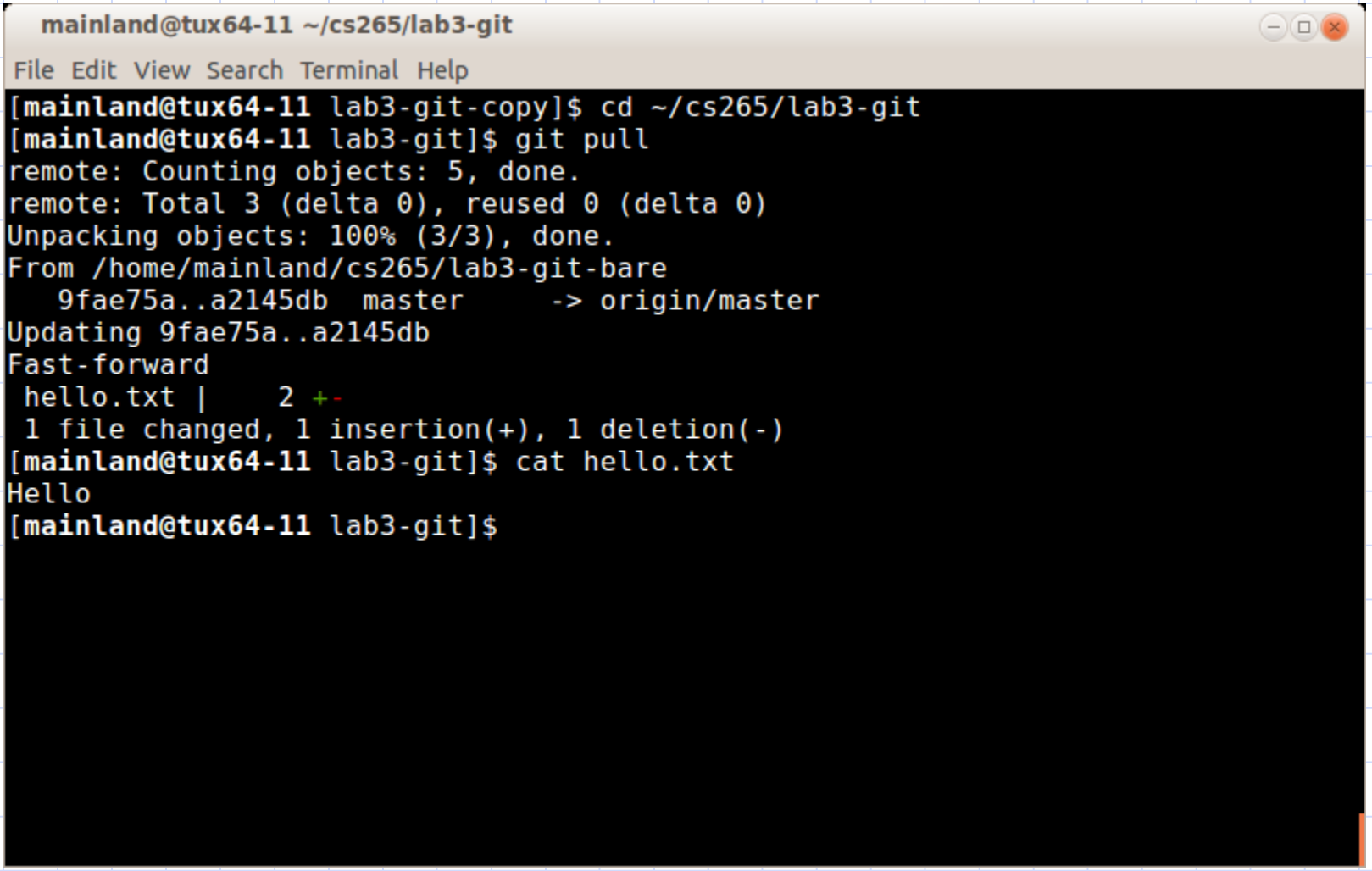
- If somebody else, e.g., wants to join the project



Pushing changes



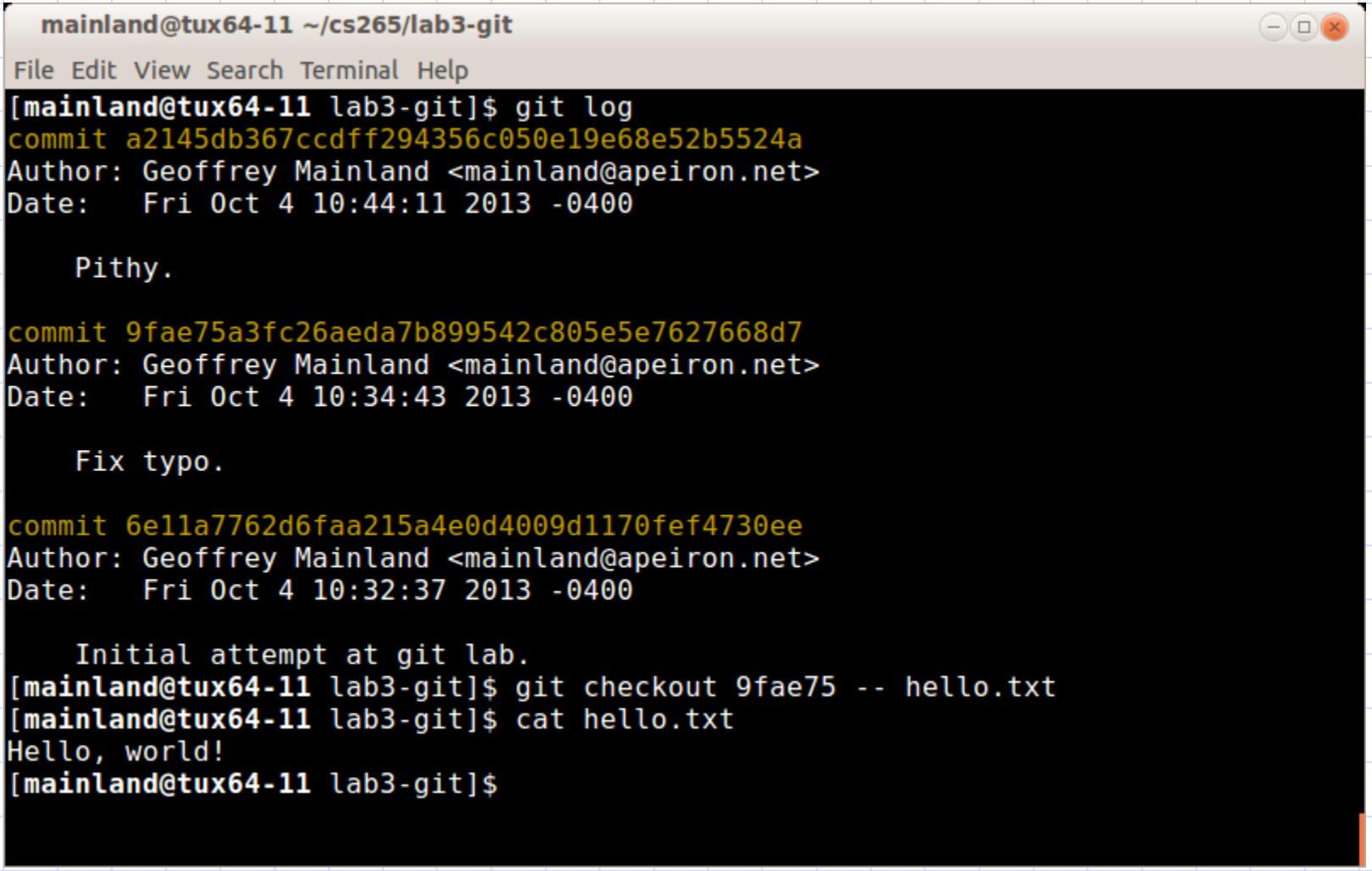
Pulling changes



Summary

Initialize a new bare repo	`git init --bare`
Add a remote repo	`git remote add`
Push changes to remote	`git push`
Pull changes from remote	`git pull`
Clone a repository	`git clone`

Checking out a specific version



Naming things in git

By branch	**`master`**
By hash	**`9fae75`**
Previous commit	**`HEAD^`**
n commits in the past	**`HEAD~`*`n`***