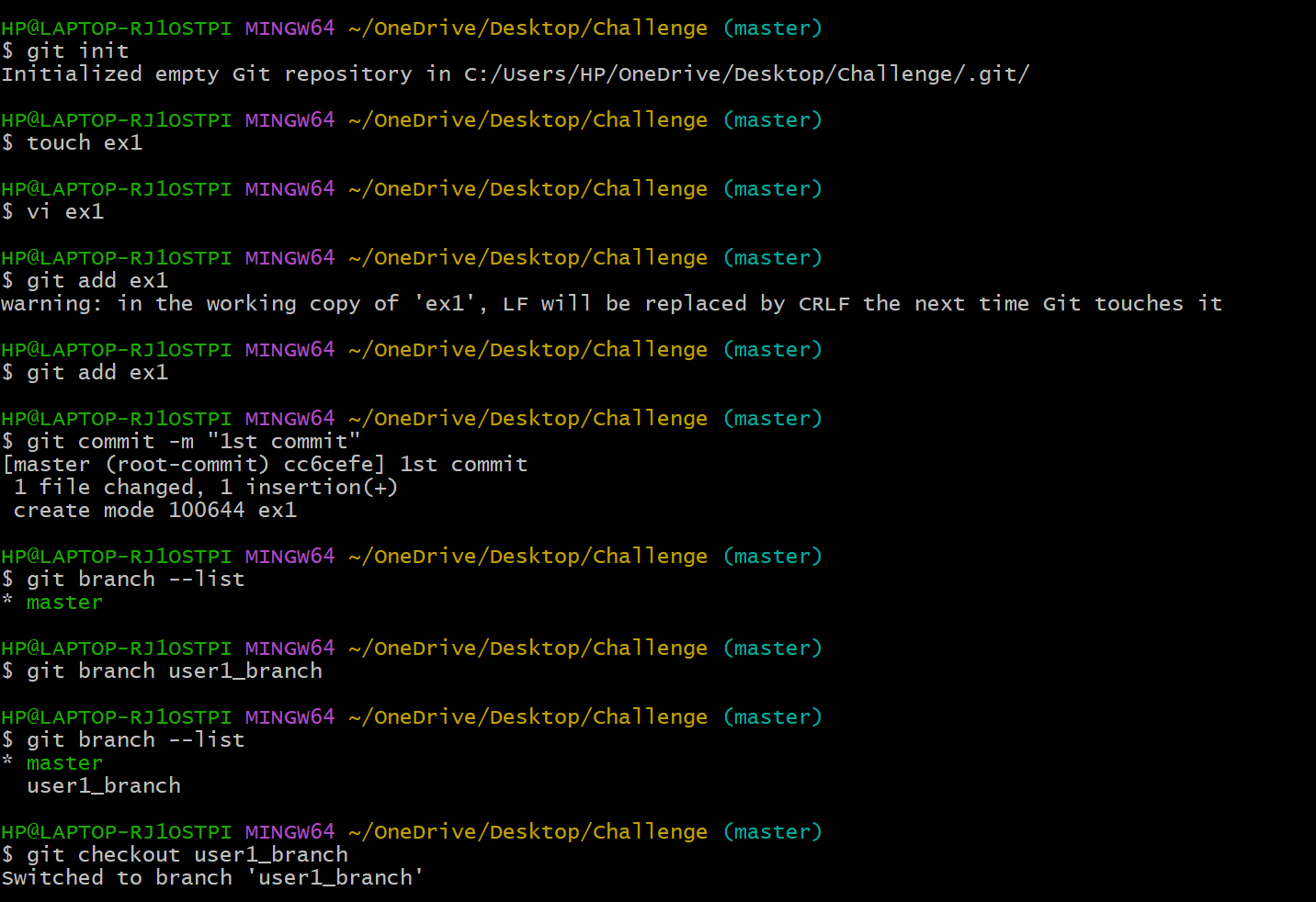
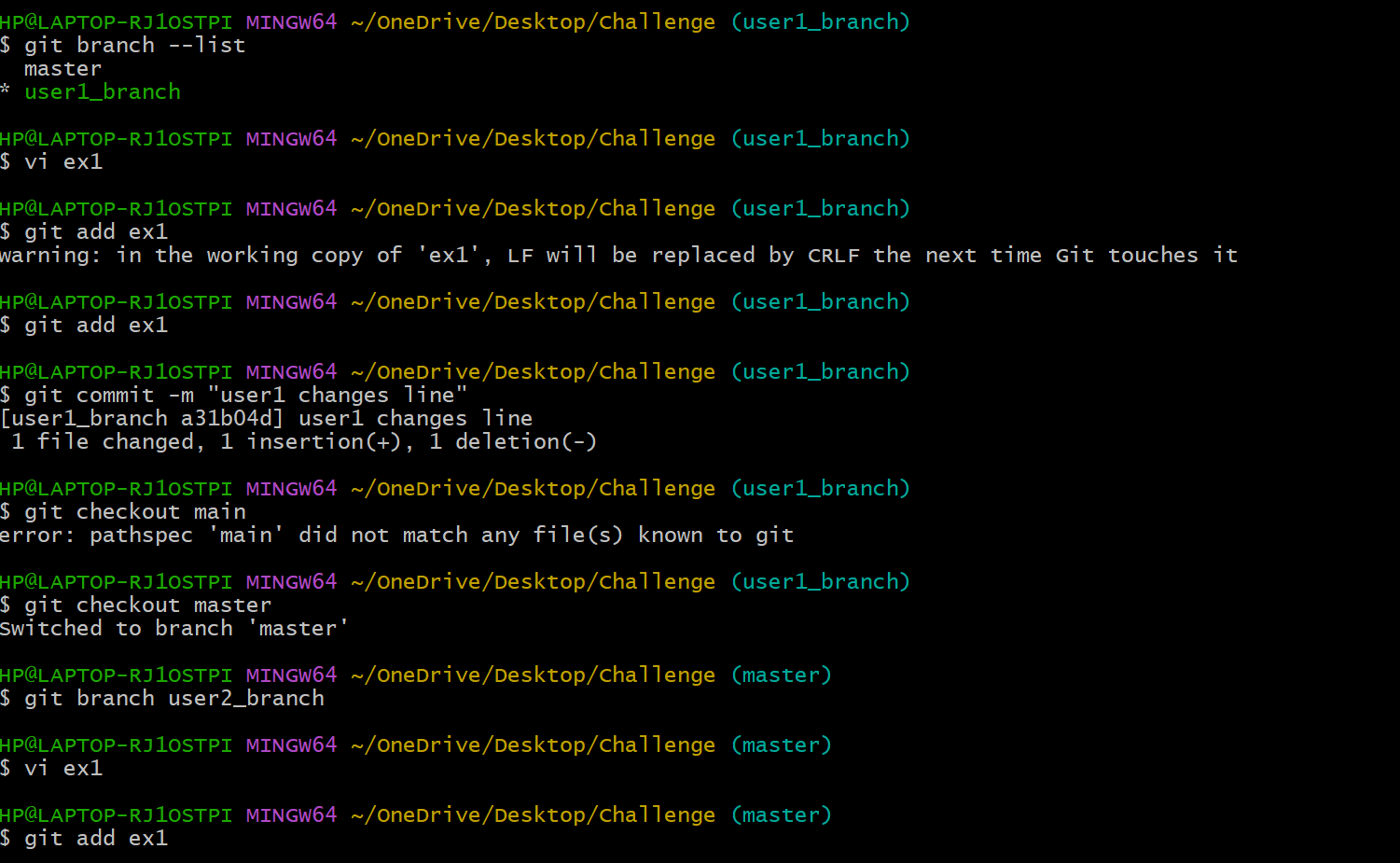
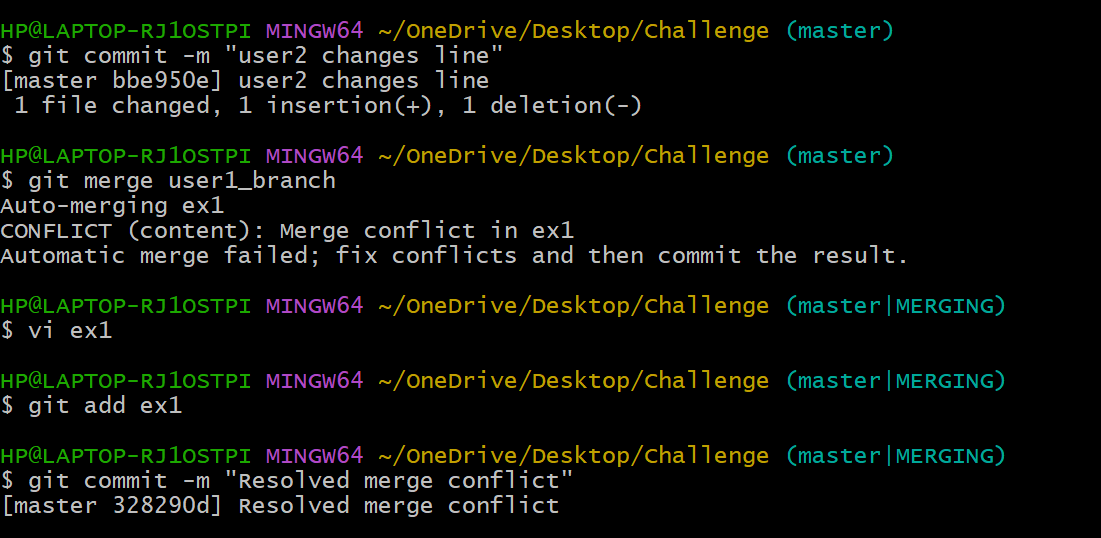
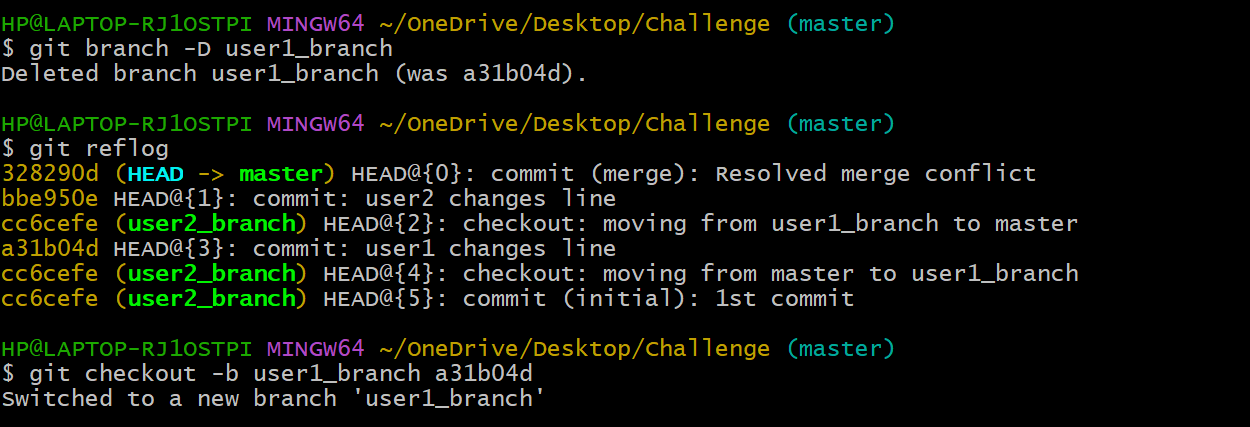
1. **Resolve Merge Conflicts**
   * **Create a merge conflict intentionally (two users editing the same line).**
   * **Resolve the conflict and push the changes.**

****

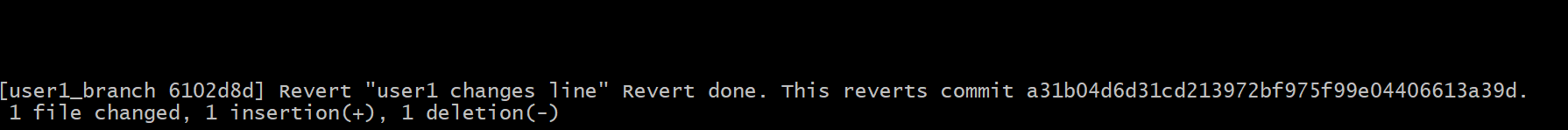
****

****

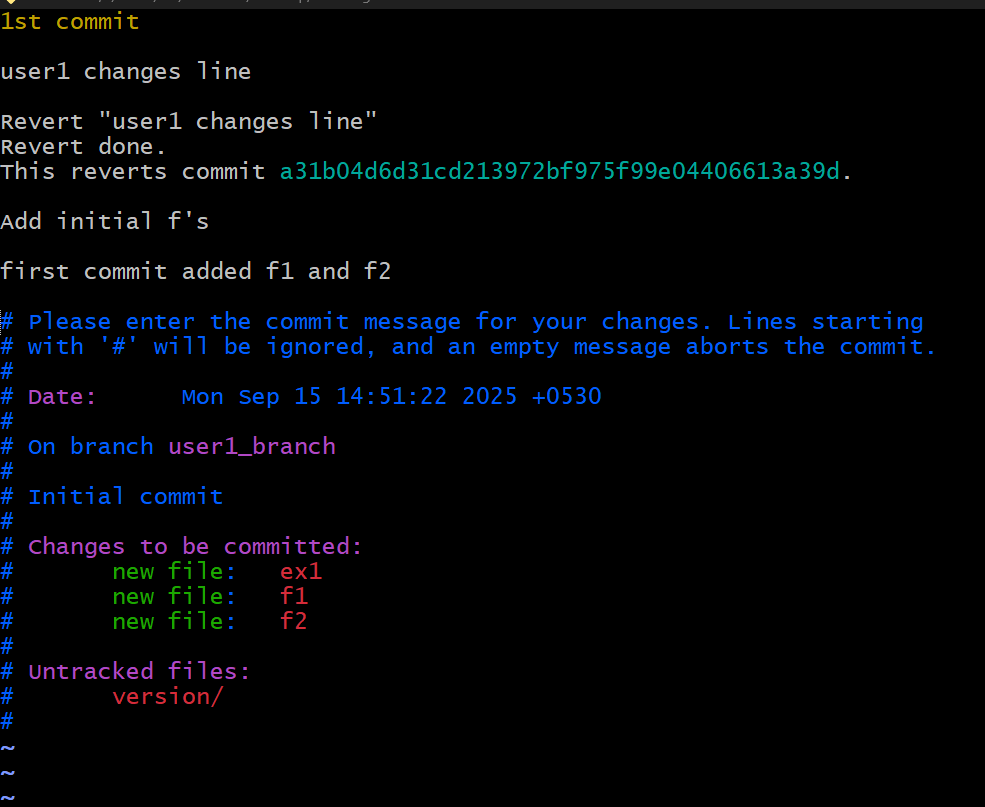
1. **Recover Deleted Branch**
   * **Delete a local branch and then recover it using the reflog.**

****

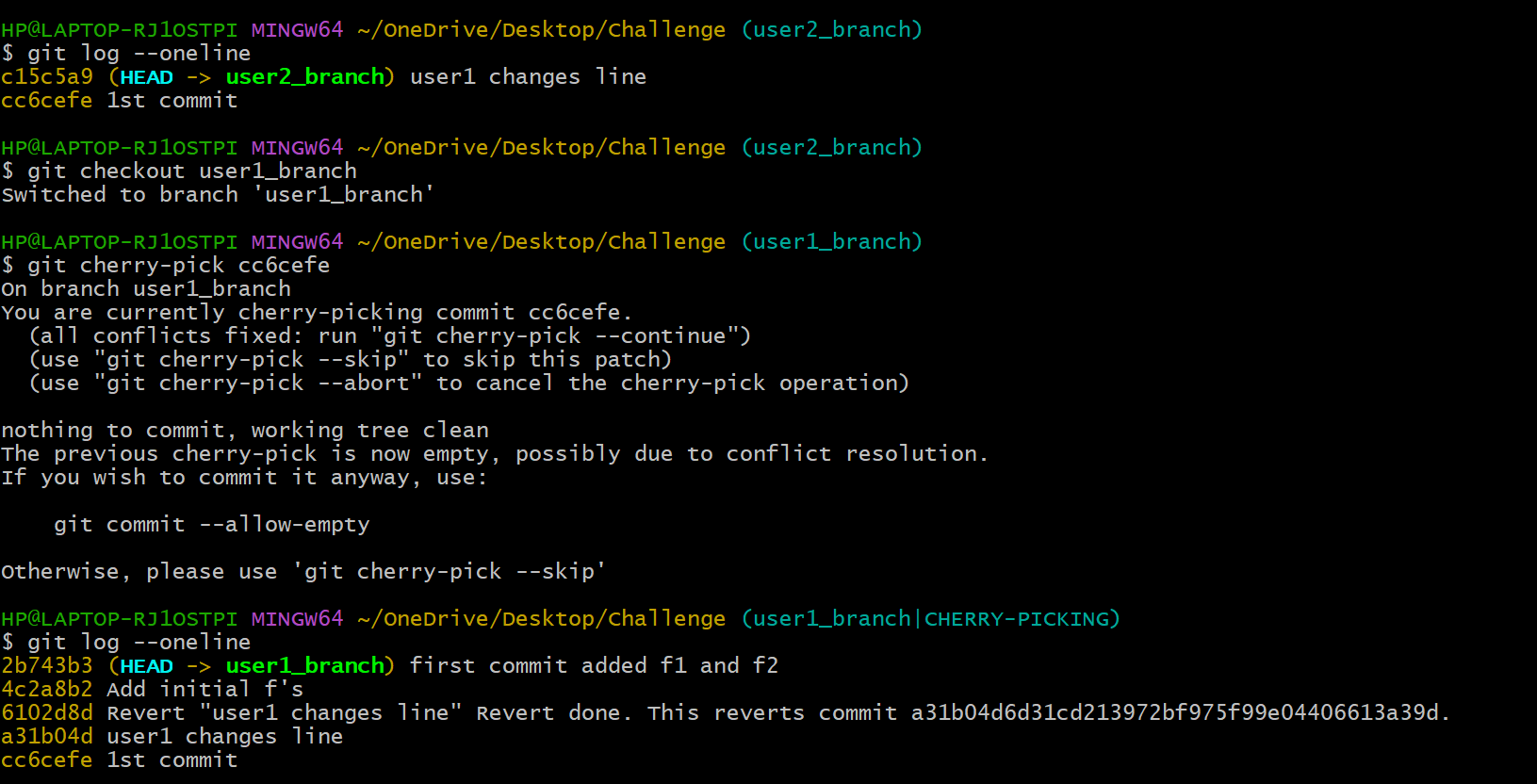
1. **Undo Wrong Push**
   * **Push a wrong commit to GitHub, then undo it without losing history.**

****

1. **Amend a Commit**
   * **Make a commit, then add a missing file to it using git commit --amend.**

****

* Use git commit -amend to edit the commit message

1. **Cherry-pick a Commit**
   * **Take a specific commit from one branch and apply it to another branch.** ****

* git log --oneline (to check the commit id and copy commit id)
* switch to other branch and enter
* git cherry-pick (commit \_id)
* git log --oneline (hence you can get the spicifc id in other branch

1. **Interactive Rebase**
   * **Reorder and squash multiple commits into a single clean commit.**

****

* git log –oneline
* git rebase -i –root
* a vi editor will open there you need to change the commit id (pick to squash) except the one you want to add in again check all log using git log

1. **Tagging & Release**
   * **Create a version tag (v1.0), push it to GitHub, then delete and restore it.**

* Use git tag -a tag\_name -m “Tag message” to create a tag
* Use git push origin tag\_name to push it to github
* Use git tag -d v1.0 to delete the tag in Git
* Use git push origin --delete tag\_name to delete tag in GitHub.

1. **Clone with Sparse Checkout**
   * **Clone only a subdirectory of a repo using sparse checkout.**

* Use git clone --no-checkout repo\_link to clone and give cd repo\_name.
* Use git sparse-checkout init --cone to enable sparse checkout   
  for current directory.
* Use git sparse-checkout add directory\_path to sparse   
  checkout configuration
* Use echo “path/to/directory” >> .git/info/sparse-checkout.
* Use git sparse-checkout set file\_name & git checkout   
  branch\_name

1. **Reset vs Revert Challenge**
   * **Demonstrate the difference between git reset --hard and git revert in a repo.**

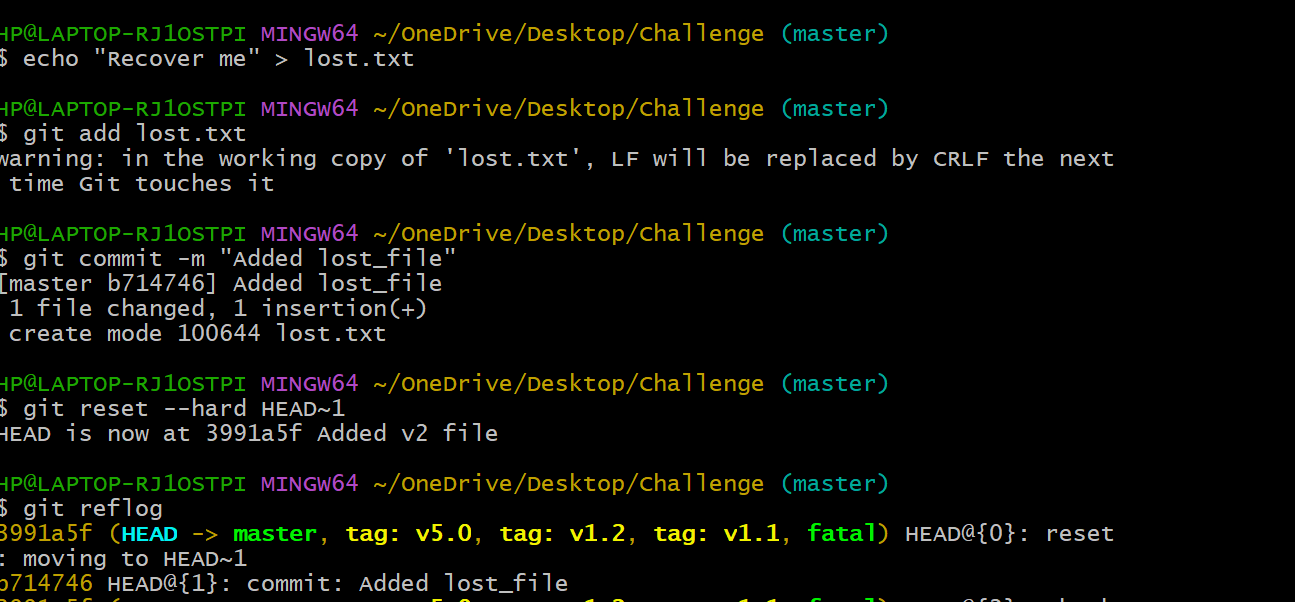
**Reset**

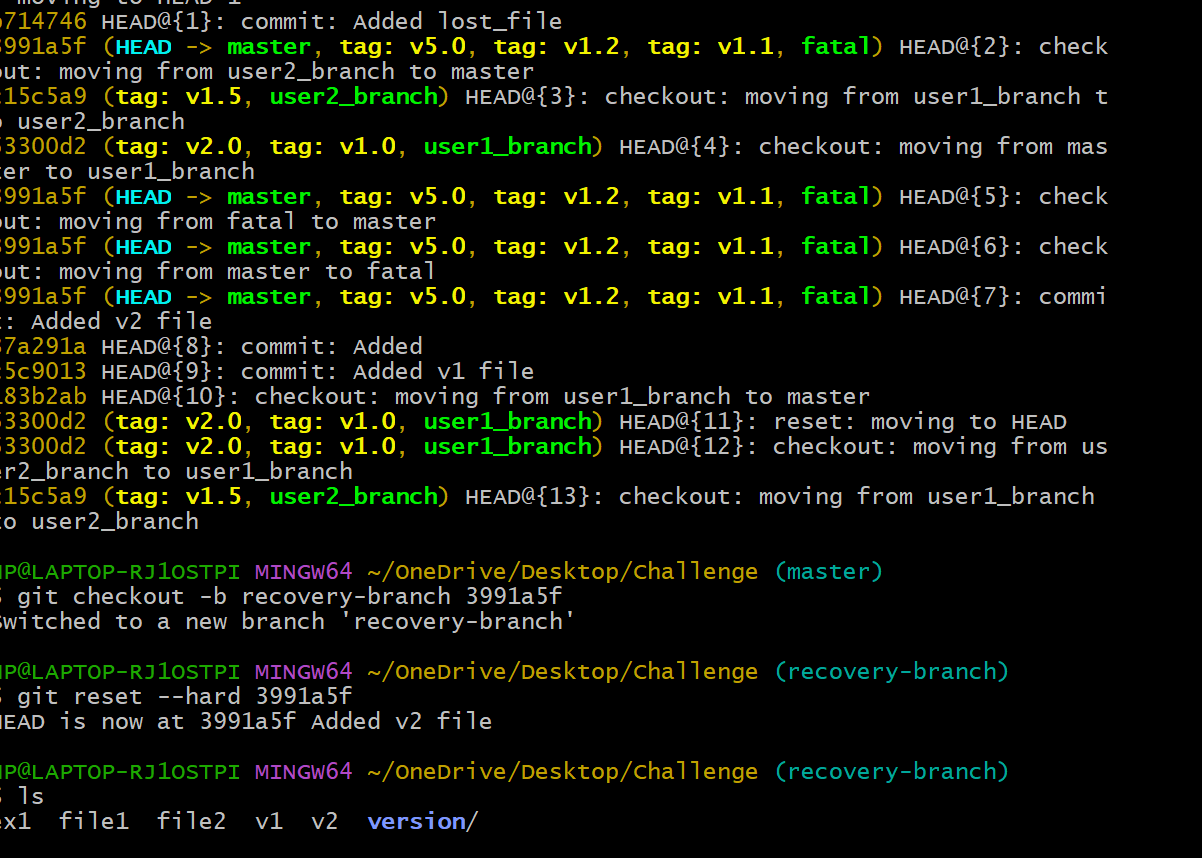
git reset --hard commit\_id will rewrites the project history. It   
moves the HEAD pointer and the current branch reference to the   
specified commit.

**Revert**

git revert commit\_id creates a new commit that undoes the   
changes introduced by the specified commit

1. **Detached HEAD Challenge**
   * **Checkout a specific commit (detached HEAD state) and create a new branch from it.**
2. **Git Hooks Challenge**
   * **Configure a pre-commit hook to reject commits without a message format (e.g., must start with JIRA-XXX).**
3. **Squash Merge vs Rebase Merge**
   * **Show the difference between squash merge and rebase merge with evidence.**
4. **Fork & Pull Request Workflow**
   * **Fork a repo, make a change, and submit a pull request to the original repo.**
5. **Recover Lost Commit**
   * **Commit something, reset hard, and then recover it using git reflog.**





* Create a file and commit it.
* Use git reset –hard HEAD~1 to revert one commit back and git reflog to list commits.
* Use git checkout -b recovery-branch commit\_id to create new branch at that commit.
* Use git reset –hard commit\_id to move head back to that commit.