ORACLE

Merge Requests and Version Management

Objectives

After completing this lesson, you should be able to:

- Describe Version Control System and the use cases of Version Control System
- Describe the fundamentals of Code Review and how to execute code review
- Describe the context of merging and how to manage merge requests with Visual Builder Studio
- Manage branching and staging with GIT and Visual Builder Studio
- Integrate GIT and GIT operations with Visual Builder Studio

Version Control Systems

1. In most organizations, different people work on the same file and there is a need to know who changed what.
2. Version control enables users to track changes to a file or files and, if something goes wrong, revert to a previous version of the file.
3. A version control system records changes to a file or set of files over time so that you can recall specific versions later.
4. There are three types of version control systems:
   - Local
   - Centralized
   - Distributed

3

Local Version Control System

The simplest form of version control is to copy files into another directory. However, files can unintentionally get overwritten and lost. To deal with this issue, local version control systems have a simple database that keeps all the changes to files under revision control.

- **Example:** RCS in MacOS Developer Tools, which works by keeping patch sets (differences, or “diffs,” between files). You can re-create what a file looked like at any point in time by adding up all the patches.
- **Advantages:** Simple
- **Disadvantages:** Error prone

Centralized Version Control Systems

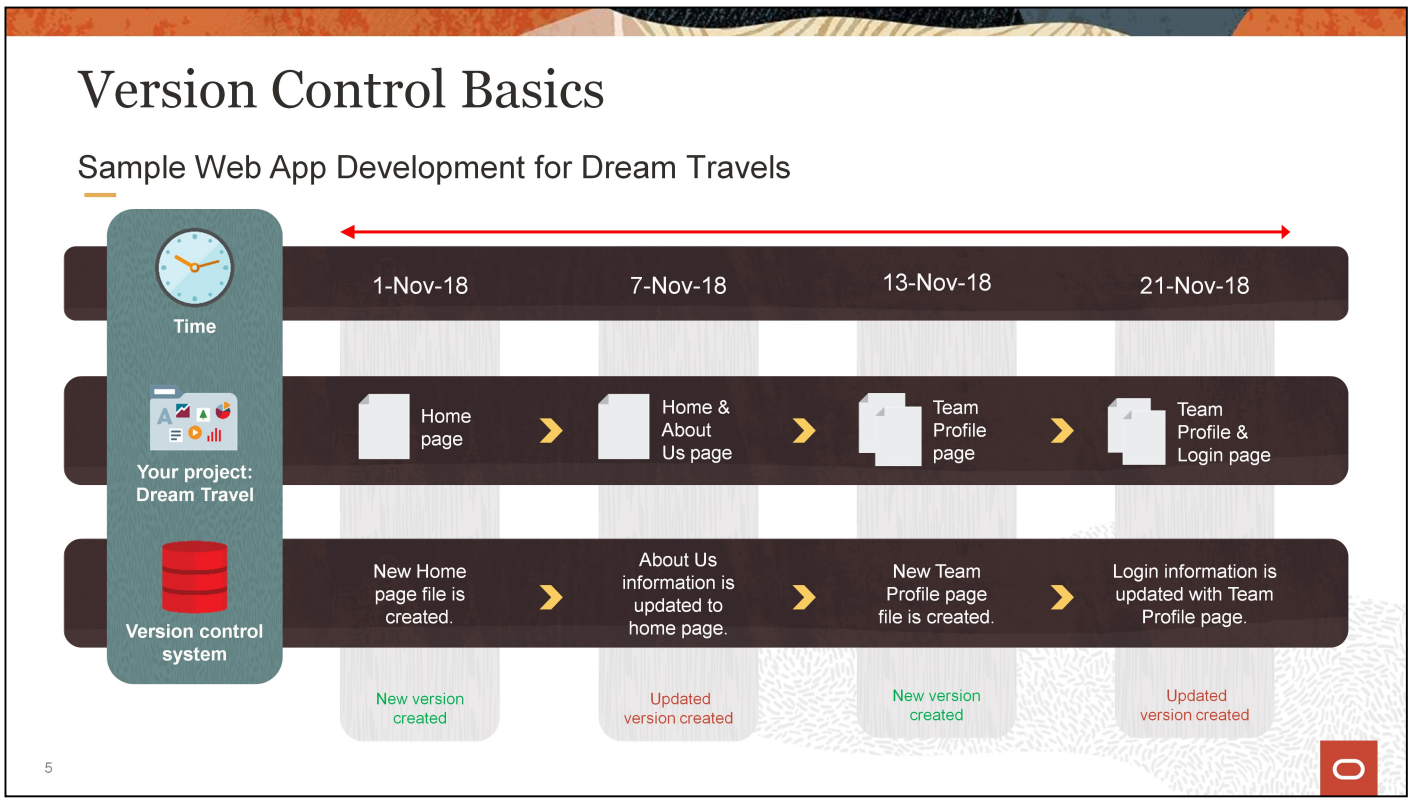
Addresses the need to collaborate with developers on other systems by using a single server that contains all the versioned files, and several clients that check out files from that central place

- **Examples:** CVS, Subversion, Perforce
- **Advantages:** Easy to administer a central database instead of several local databases
- **Disadvantages:** Single point of failure if the database goes down, get corrupted, and so on

Distributed Version Control Systems

Clients do not just check out the latest snapshot of the files; they fully mirror the repository. If a server being collaborated with dies, any client repository can be copied back up to the server to restore it. Each clone is a full backup of all of the data.

- **Examples:** Git, Mercurial
- **Advantages:** Enables developers to work with several remote repositories. Developers can collaborate with different groups of people in different ways simultaneously within the same project. Workflows that are not possible in centralized systems, such as hierarchical models, can be used.



A version control system keeps the history of changes over a file or set of files.

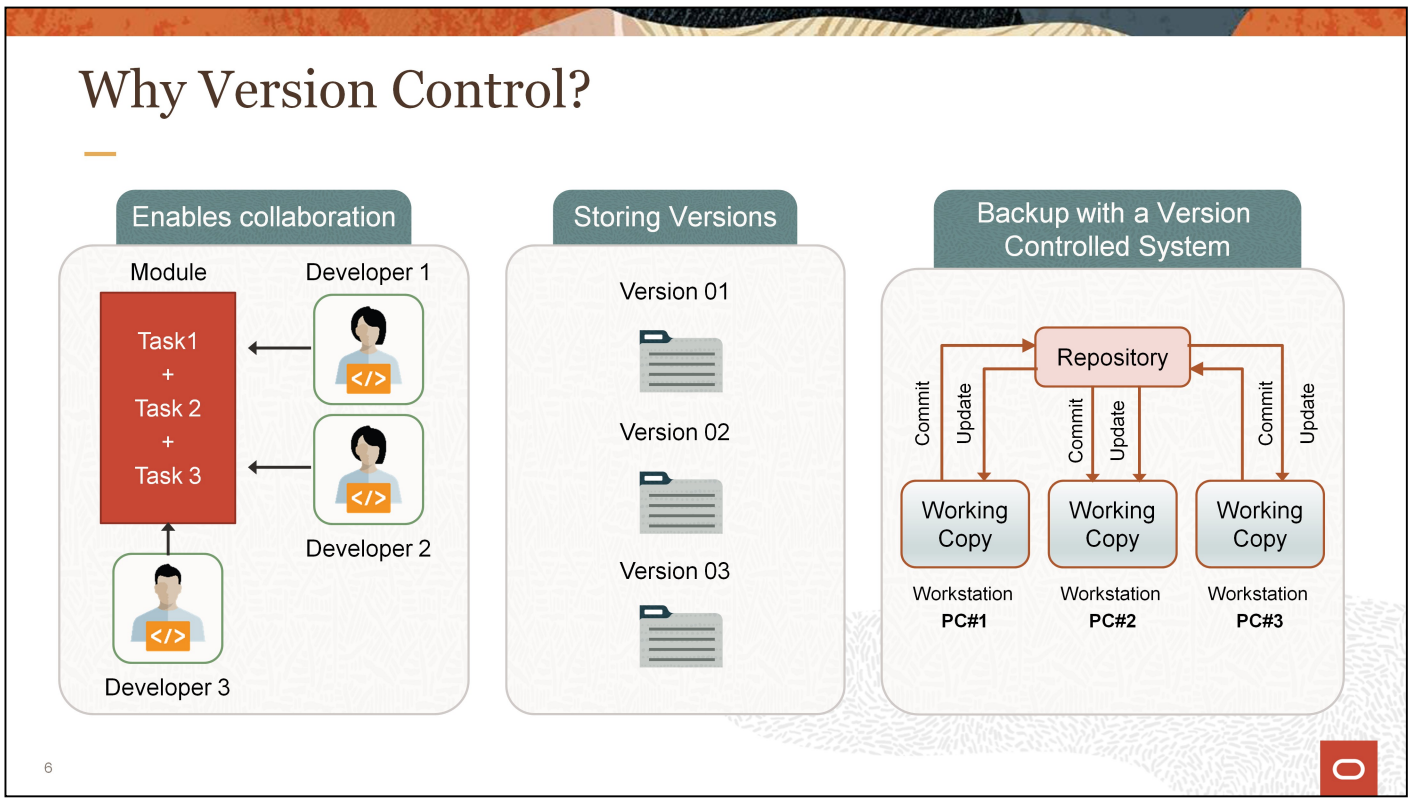
If you wanted to keep track of the changes, without a version control system you would need to store multiple copies of the same file for each version that you wanted to keep. To clean up, you would need to archive such files or remove older versions, which would result in a loss history.

Collaborating in a file that has been modified by multiple individuals means trying to find changes made to a file by comparing the same file to other versions and merging all changes into a final file.

Version control systems avoid this problem by handling file history and merging changes, if necessary. Individual changes to files are stored in the version control, and a file can be reconstructed to other versions by applying each change.

If two developers modify a file, both changes are applied and if they do not conflict with each other, they are accepted immediately. A change conflict occurs if the developers modify, say, the same line of code. In this case, the user trying to apply the change has to decide which change will stay in the file by doing a manual merge.

With version control, you decide which files to track and which files to ignore.



Enables Collaboration: Collaboration is possible among all the developers involved in the same module. It enables your project to evolve as a whole from the start and saves time by removing confusion among developers.

Storing Versions

- Snapshots of all the versions are properly documented and stored.
- Versions are also named accurately.

Backup: If your server/repository crashes, a backup is always available in your working copy.

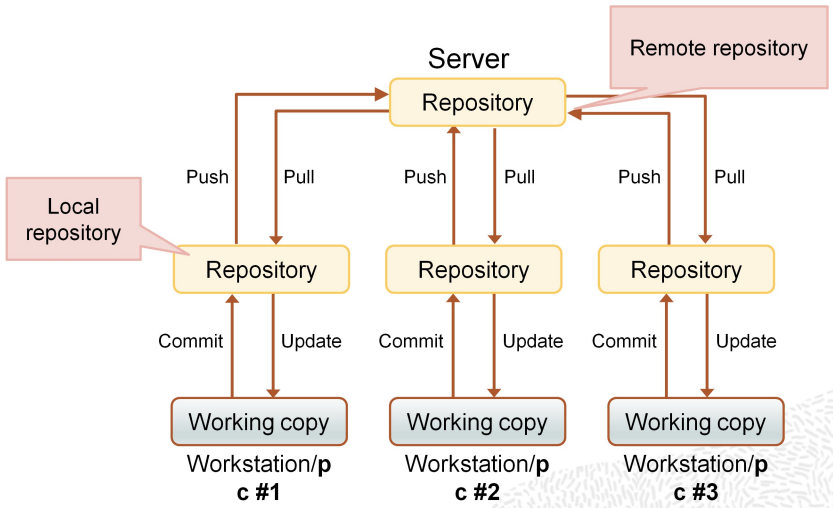
Features of Git

When you make any modification in your project, the version control system provides you a proper description of:

- What exactly was changed
- When it was changed

Thus, you can analyze how your project evolved between versions.

Distributed Version Control System



Note: A repository is a data space where all files related to a project are stored.

7

1. Initially, each developer clones the remote Git repository to his/her computer and then makes changes to the local Git repository.
2. They then transfer or push those changes to the remote repository.
3. They update their local repository by pulling or transferring information from the central repository.

Version Management

- Collaborative development on source code
- Track changes
- Manage branches and releases
- Allow rollback of changes

Git

1. Git is a free and open source, distributed version control tool for tracking changes in digital files (project-related source code documents), and coordinating work on those files among multiple people.
2. There are several options available to achieve version control.
3. Git is a very popular and successful one, due to its **distributed** nature.
   - Git is locally enabled, so connectivity issues do not hinder your work.
   - users do not need any complicated server setup or software; a simple command-line tool is enough to work with Git.

9

Nearly every operation local: Most operations in Git need only local files and resources to operate. Generally, no information is needed from another computer on your network.

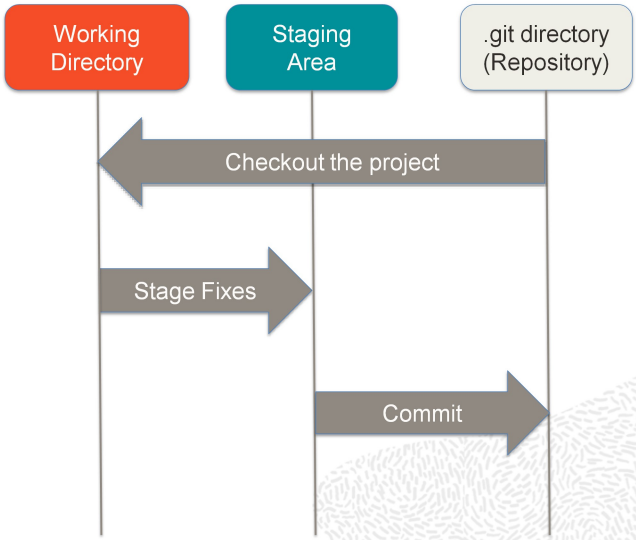
Integrity: Everything in Git is check-summed before it is stored and is then referred to by that checksum. This means it is impossible to change the contents of any file or directory without Git knowing about it.

Adds only data: When you perform actions on Git, nearly all of them add data to only the Git database. It is difficult to get the system to do anything that is not undoable or to make it erase data in any way.

States of Files in Git

1. **Modified:** users have made changes to the file but have not yet committed them to the local repository.
2. **Staged:** users have marked a file in its current version to go into your next commit snapshot.
3. **Committed:** The data is safely stored in your local repository.

Sections of a Git Project



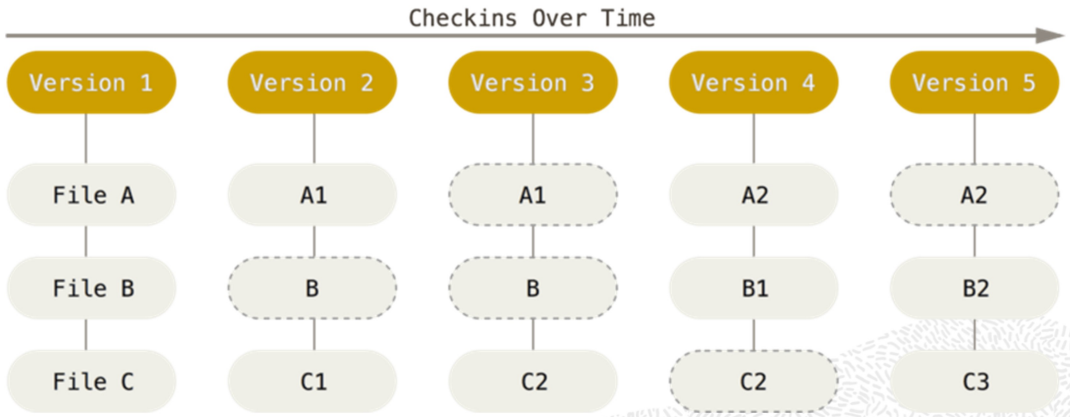
11

Git Directory: Contains metadata and the object database for you project or what is copied when you clone a repository from another computer

Working Tree: A single checkout of one version of the project, where files are pulled from the compressed database in the Git directory and put on disk to use or modify

Staging Area: A file that stores information about what will go into your next commit

Git Versioning



12

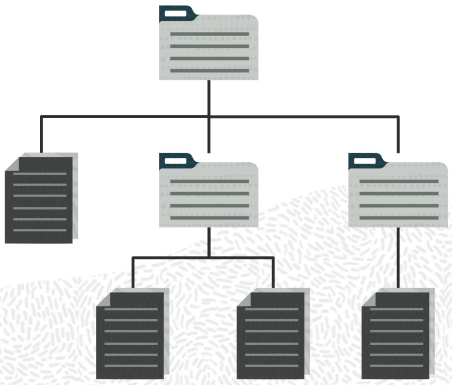
- Every time you commit, or save the state of your project in Git, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot.
- To be efficient, if files have not changed, Git does not store the file again, but just a link to the previous identical file it has already stored.
- Git treats data like a **stream of snapshots.**

Source Code

Contains:

- Directories
- Files

The substance of a software configuration

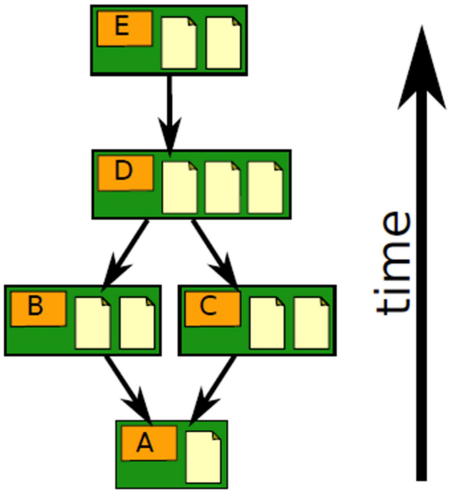


13

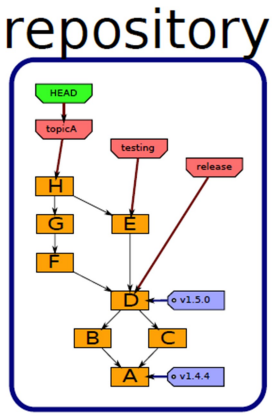
Repository

Contains:

- Files
- Commits
- Ancestry relationships – *which form a directed acyclic graph (DAG) over time*



Putting It All Together – 1

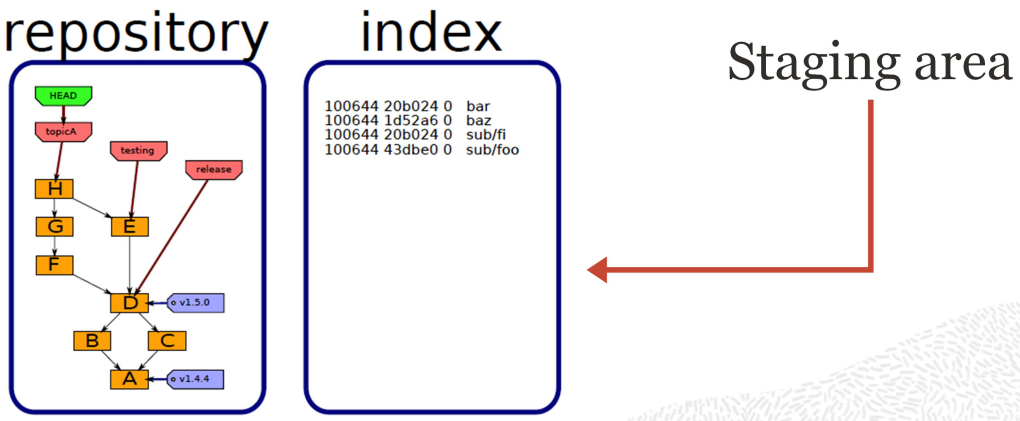


History

Stored in the ".git" directory

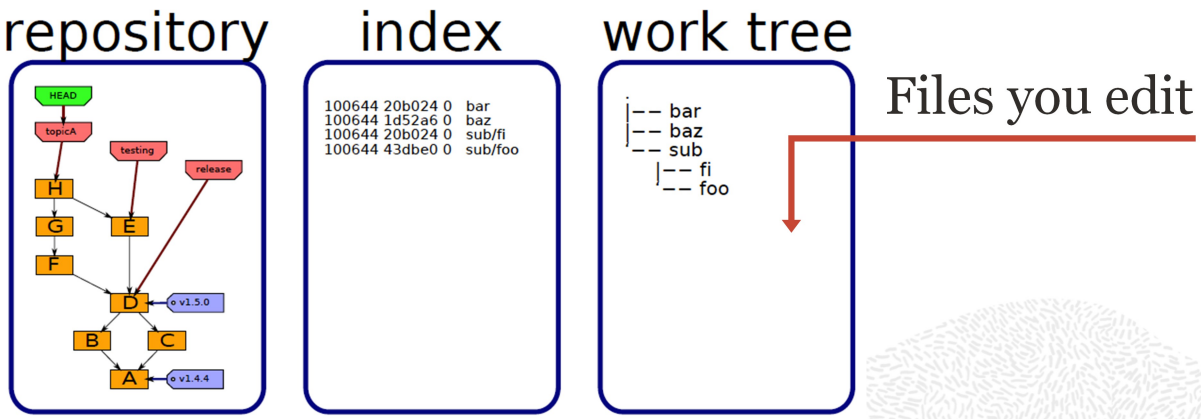
15

Putting It All Together – 2



Also stored in ".git" directory

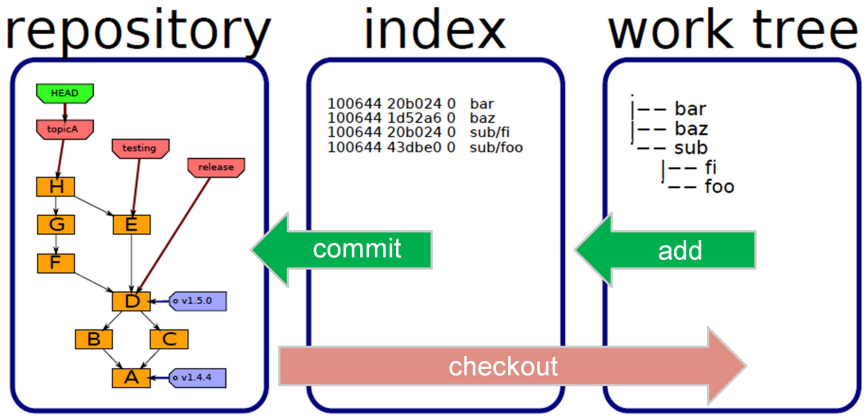
Putting It All Together – 3



Stored in the parent directory of the ".git" directory

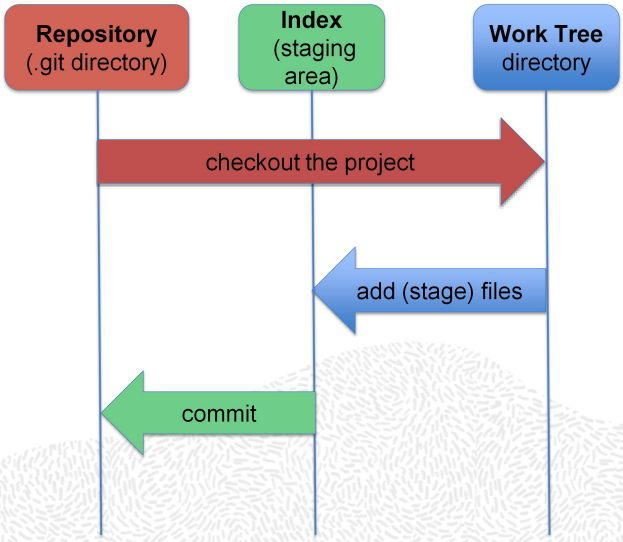
17

Putting It All Together – 4



Git Basic Operations & Commands

- Setup and Branch Management
  - **init**, **checkout**, **branch**
- Modify
  - **add**, **delete**, **rename**, **commit**
- Get Information
  - **status**, **diff**, **log**
- Create Reference Points
  - **tag**, **branch**



Basic Git Operations

Command	Operation
`git init`	Create a new repository in an existing project or directory.
`git clone`	Clone an existing repository from another server.
`git add`	Start file tracking and stage changes.
`git commit`	Commit changes.
`git pull`	Pull changes from a remote server and merge with a file in a working directory.
`git push`	Push local changes to a remote server.
`git status`	View changes since the last commit.
`git rm`	Stop file tracking.
`git mv`	Rename a file.

Common Git Commands

CLONING A REPOSITORY

```
$ GIT CLONE HTTPS://GITHUB.COM/DBRGN/FAHRPLAN
```

ADD NEW FILE

```
$ git add README.rst
```

REMOVE FILE

```
$ git rm file.py
```

COMMIT CHANGES

```
$ git commit -am 'First commit'
```

21

Other Git Commands

SHOW LOG

```
$ git log
```

SHOW COMMITS

```
$ git show
```

SHOW DIFFS

```
$ git diff
```

UNMODIFY MODIFIED FILE

```
$ git checkout -- file.py
```

REVERT A COMMIT

```
$ git revert 1776f5
```

22