19

Introduction to Git

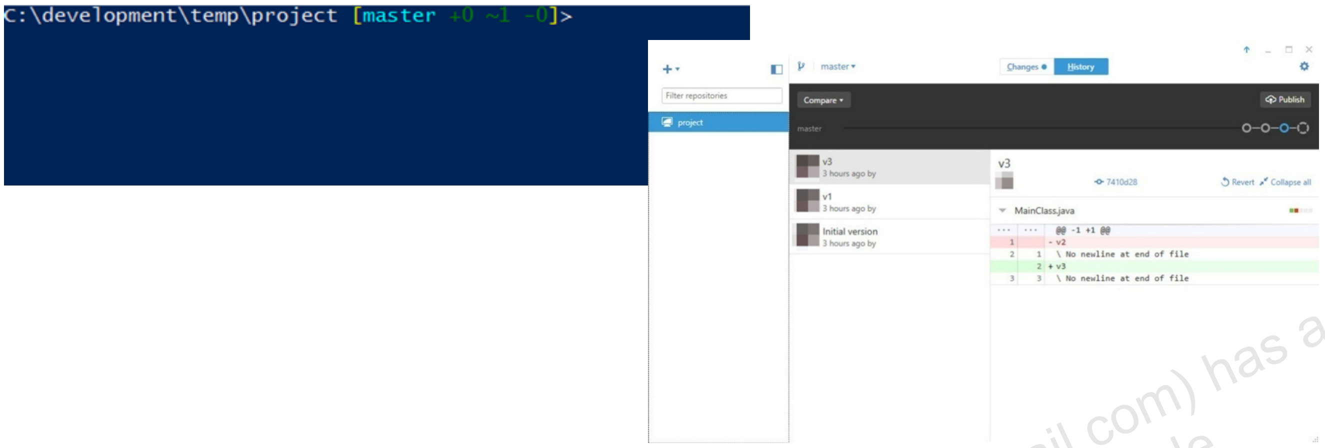
Topics

- Introduction to Git
- Performing basic Git operations

Getting Started with Git

Git can be used from the command line or with a GUI.

Understanding the Git command line commands is important to understanding Git. This is why this course teaches you how to use Git from the command line.



```
C:\development\temp\project [master +0 ~1 -0]>
```

```
v3
3 hours ago by
v1
3 hours ago by
Initial version
3 hours ago by
```

```
v3
7410d28
MainClass.java
...  ...  @@ -1 +1 @@
 1        - v2
 2    1   \ No newline at end of file
      2   + v3
 3    3   \ No newline at end of file
```

Learning the basic commands help you use the graphic clients as the operations found in them have the same name as their command line counterparts.

States of Files in Git

In Git, your files can be in one of the following states:

- **Committed:** The data is safely stored in your local database.
- **Modified:** You have made changes to the file but have not committed it to your database yet.
- **Staged:** You have marked the file in its current version to go into your next commit snapshot.

Sections of a Git Project

There are three main sections of a Git project:

- **Git directory:** Contains metadata and the object database for your project or what is copied when you clone a repository from another computer
- **Working directory:** 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

ORACLE

Topics

- Introduction to Git
- Performing basic Git operations

Basic Git Operations

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

These are the basic operations that you must become familiar with to use Git in this course.

Most Git operations can be done using a `.gitignore` file, which tells Git to avoid putting certain files into the repository, and this relatively small set of commands.

Using the `git init` Command

- Create a new Git repository in an existing project or directory:

```
$ git init
```

This creates a `.git` subdirectory that contains the repository files in the project directory, but nothing in your project is tracked yet.

The `git add` and `git commit` commands are described in subsequent slides.

After Setting Up a Local Repository

- Next, specify files to track and commit them:

  ```
  $ git add .
  $ git commit –m "<commit-message>" .
  ```

  Now you have a Git repository with tracked files and an initial commit.
- Reference page: https://git-scm.com/docs/git-clone

Using the `git add` Command

- Begin tracking a new file or all files in a directory:

```
$ git add .
```

- Accepts a path name for a file or for a directory
- If the path is for a directory, the command adds all files and all subdirectories in that directory
- Git stages a file exactly as it is when `git add` is run. If the file is modified after the command is run, the command must be rerun to stage those modifications.
- Use `git status` to list files that have changes staged for the next commit.
- Reference page: https://git-scm.com/docs/git-add

The `git add` command updates the "index" using the current content found in the working tree, to prepare the content staged for the next commit. The index contains a snapshot of the content of the working tree. This snapshot is taken as the content of the next commit. After making changes to the working tree, and before running the commit command, use the add command to add new or modified files to the index.

This command can be used multiple times before a commit. It only adds the content of the specified file or files at the time the add command is run. To include subsequent changes in the next commit, run `git add` again to add the new content to the index.

Use the `git status` command to get a list of changed files that are staged for the next commit.

Using the `git commit` Command

- Store the current contents of the index in a new commit with a message describing the changes:

```
$ git commit -m "commit message" .
```

- Use `-a` to automatically add changes from all tracked files and remove files in the index that have been removed from the working tree, and then commit.
- Use `--dry-run` to see a summary of what would be included for the next commit without actually doing the commit.
- Use `-m "<msg>"` as the commit message.
- If you make a commit and then find a mistake, recover from it with `git reset`.
- Reference page: https://git-scm.com/docs/git-commit

Content to be added to a commit can be specified in several ways:

- Use `git add` to incrementally add changes to the index before using the `commit` command.
- List files as arguments to the `commit` command, in which case the commit will ignore changes staged in the index, and instead record the current content of the listed files.
- Use `git rm` to remove files from the working tree and the index before using the `commit` command.

If you specify a message with the `-m` option, that message will be used as the commit message. If multiple `-m` options are given, their values are concatenated as separate paragraphs. If you do not specify a commit message, Git will start your default editor and prompt you to type in a message.

Using the `git status` Command

- Shows the status of the working tree:

```
$ git status
```

- Answers two questions:
  - What have you changed but not yet staged?
  - What have you staged that you are about to commit?
- Use the `-s` or `-short` option to see input in short-format.
- Use the `-v` or `-verbose` option (or the `git diff` command) to also see file level changes.
- Reference page: https://git-scm.com/docs/git-status

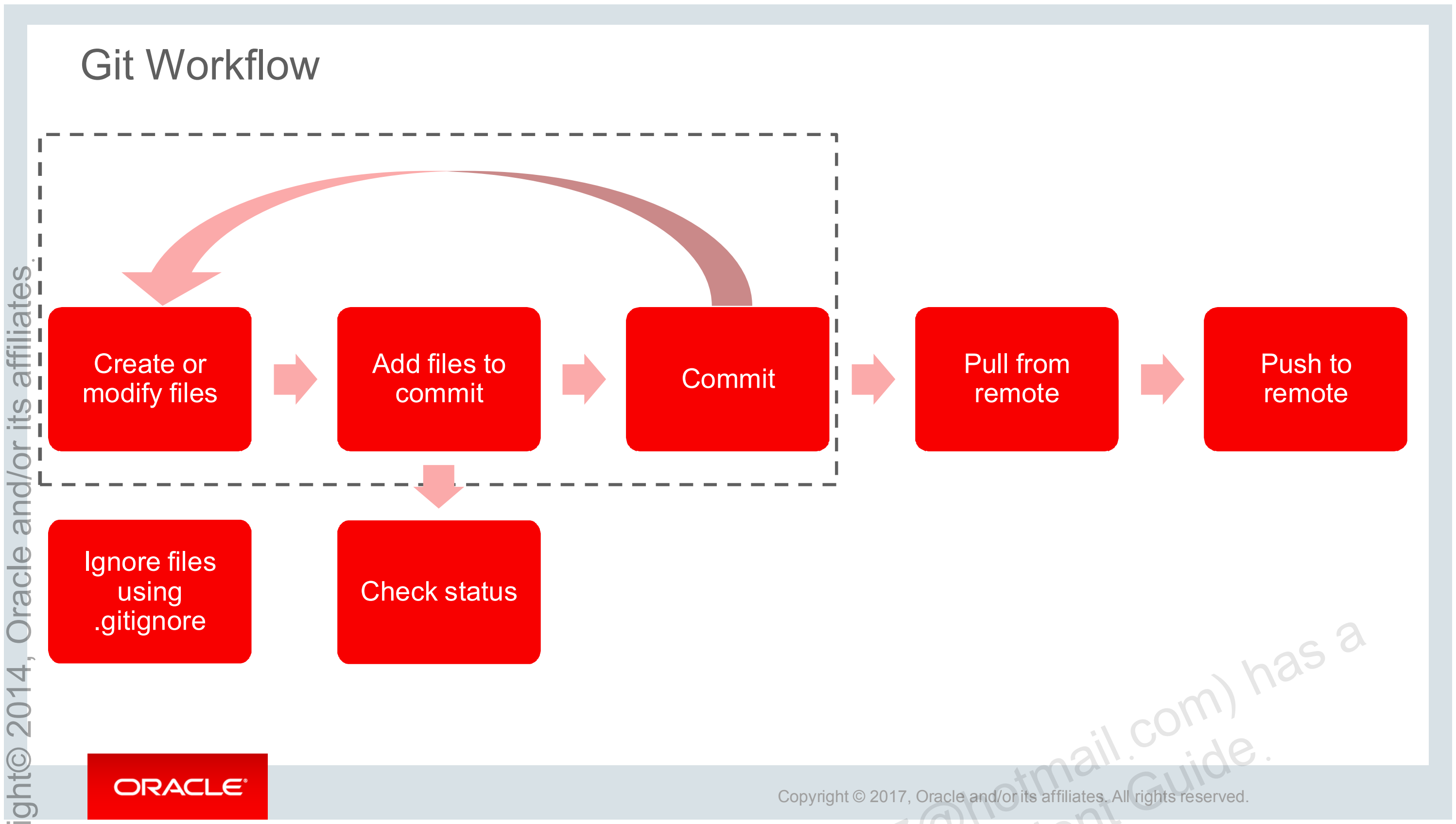
Each file in your working directory can be in one of two states: tracked or untracked.

- Tracked files are files that were included in the last snapshot. These files can be unmodified, modified, or staged.
- Untracked files include any files in your working directory that were not in your last snapshot and are not in your staging area.

Output from the `git status` command shows what you *would* commit by running `git commit` as well as what you *could* commit by running `git add` before running `git commit`.

Standard output from the command is quite verbose and self explanatory. If you opt to use the short-format, there are several things to be aware of:

- New untracked files have ?? next to them.
- New files that have been added to the staging area have an A next to them.
- Modified files have an M next to them.
- The output has two columns: the left hand column indicates that the file is staged, while the right hand column indicates that the file is modified



Git Workflow

To work with an initialized repository, either by cloning or initializing an empty folder, you follow the workflow in the slide.

You modify or create files and folders freely, and if you want to track changes to that file, add the files to the commit using the `add` command.

When you want to store a copy of your files as a new version in Git, use the `commit` command and all the folders added will be committed.

The `add` command adds the file to the commit, not to the file system, and Git is smart enough to figure if a file is new or has been modified.

You then repeat the cycle of creating and modifying files, adding to the commit, and committing them to Git.

Ignoring Files That Should Not Be Committed

- A `.gitignore` file tells Git which files and directories to ignore, before you make a commit.
- The `.gitignore` file must reside in the working directory, not in the `.git` directory.
- Use patterns to delineate files to ignore, rather than explicitly listing files and directories.
- List all ignored files in this project:

```
$ git ls-files --other --ignored --exclude-standard
```

- See https://github.com/github/gitignore for good `.gitignore` file examples for different projects and languages.

Git uses the `.gitignore` file to determine which files and directories to ignore, before you make a commit. To suppress versioning of files and paths, add the patterns in the file you want Git to ignore. Put the `.gitignore` file in the working directory. Do not put the `.gitignore` file in the `.git` directory.

Setting up a `.gitignore` file before you get going is generally a good idea so you do not accidentally commit files that you really don't want in your Git repository.

Commit a `.gitignore` file into your repository to share the ignore rules with other users that clone the repository.

The rules for the patterns you can put in the `.gitignore` file are as follows:

- Blank lines or lines starting with # are ignored
- Use standard glob patterns
- Start patterns with a forward slash (/) to avoid recursivity
- End patterns with a forward slash (/) to specify a directory
- Negate a pattern by starting it with an exclamation point (!)

Removing Files from the Repository

- Remove files from the working tree and from the index:

```
$ git rm [file]
```

- Will not remove modified files that have been staged, unless the removal is forced with `–f`
- Does not remove the file just from the working directory
- Reference page: https://git-scm.com/docs/git-rm

The `git rm` command removes tracked files from the working directory and from the index. If you simply remove the file from your working directory, it will still show up under the "Changed but not updated" area when you run the git status command. After using the `git rm` command and then performing a `git commit`, the removed file will also disappear from your working directory.

Renaming Files in Git

- Move or rename a file, a directory, or a symlink:

```
$ git mv [old-file-name] [new-file-name]
```

- Equivalent of doing these three commands:

```
$ mv [old-file-name] [new-file-name]
$ git rm [old-file-name]
$ git add [new-file-name]
```

  The renaming is finalized at the next commit.

- Reference page: https://git-scm.com/docs/git-mv

The `git mv` command is one command instead of three. It is a convenience function.

Using the `git clone` Command

- Sets up your *local* master branch to track the *remote* master branch on the server you cloned from.
- Get a copy of an existing Git repository:

```
$ git clone [url]
```

- Get a copy of an existing Git repository with a name that is different from the original repository name:

```
$ git clone [url] [new-repository-name]
```

- A cloned remote repository is automatically added under the name “origin.”
- Reference page: https://git-scm.com/docs/git-clone

Instead of getting just a working copy, Git receives a full copy of nearly all data that the server has: a directory with the same name as the directory in the specified [url], a .git directory inside it, all the data for that repository, and a checked out working copy of the latest version. If you want to clone the repository into a directory named something other than the original name, you can specify that as the next command-line option:

$ git [url] [new-directory-name]

Git has several different transfer protocols that you can use: https:// protocol, git://, file://, or user@server:path/to/repo.git that use the SSH transfer protocol.

Using the `git pull` Command

- Fetches data from the server you cloned, and automatically tries to merge files into the code that you are currently working on
- Fetch from and integrate with another repository or a local branch:

```
$ git pull
```

- Contrast with `git fetch`, which does not automatically merge the data that is pulled down from the origin server with any of your work, or modify what you are currently working on; requires manually merging into your work when you are ready.
- Reference page: https://git-scm.com/docs/git-pull

Using the `git push` Command

- Push your changes upstream to a remote server and update remote refs and associated objects :

  ```
  $ git push [remote-name] [branch-name]
  ```

- If you cloned your repository, use this command to push your master branch and all its commits to your origin server:

  ```
  $ git push origin master
  ```

- Proactively do a `git pull` before doing a `git push` to avoid most conflicts.
- Reference page for git push: https://git-scm.com/docs/git-push

```
$ git push origin master
```

This command works only if:

- You cloned from a server that you have permission to write to
- Nobody has pushed their changes in the meantime

If you and someone else clone at the same time and they push upstream before you do, your push will be rejected. You will have to pull down their work first and incorporate it into yours before you will be allowed to push.

Summary

In this lesson, you should have learned how to:

- Explain the characteristics that make Git an optimal version control system
- Describe how to use basic Git commands to create or clone a repository, add files to be tracked, commit changes, pull from, and push to a remote repository