GIT Advance Commands

VISHWANATH M S

VISHWACLOUDLAB.COM

Agenda

- Git ignore
- Git stash
- Git revert
- Git Rebase

GIT Ignore -- Lets explore

Git ignore

- Git sees every file in your working copy as one of three things
  - tracked - a file which has been previously staged or committed;
  - untracked - a file which *has not* been staged or committed; or
  - ignored - a file which Git has been explicitly told to ignore.

Git ignore – NOT TO BE commited

- dependency caches, such as the contents of /node_modulesor /packages
- compiled code, such as .o, .pyc, and .class files
- build output directories, such as /bin, /out, or /target
- files generated at runtime, such as .log, .lock, or .tmp
- hidden system files, such as .DS_Store or Thumbs.db
- personal IDE config files, such as .idea/workspace.xml

Git ignore

- Ignored files are tracked in a special file named .gitignore that is checked in at the root of your repository.
- There is no explicit git ignore command: instead the .gitignore file must be edited and committed by hand when you have new files that you wish to ignore.
- .gitignore files contain patterns that are matched against file names in your repository to determine whether or not they should be ignored

Git ignore patterns

- Versions are maintained to hold a single source of Application.
- A system that keeps records of your changes.
- Using Centralized single source code, **Operations can access the same code what they plan to release**.
- Allows you to know who made what changes and when!!
- Easy to **Rollout** the faulty snippet of code or complete release.

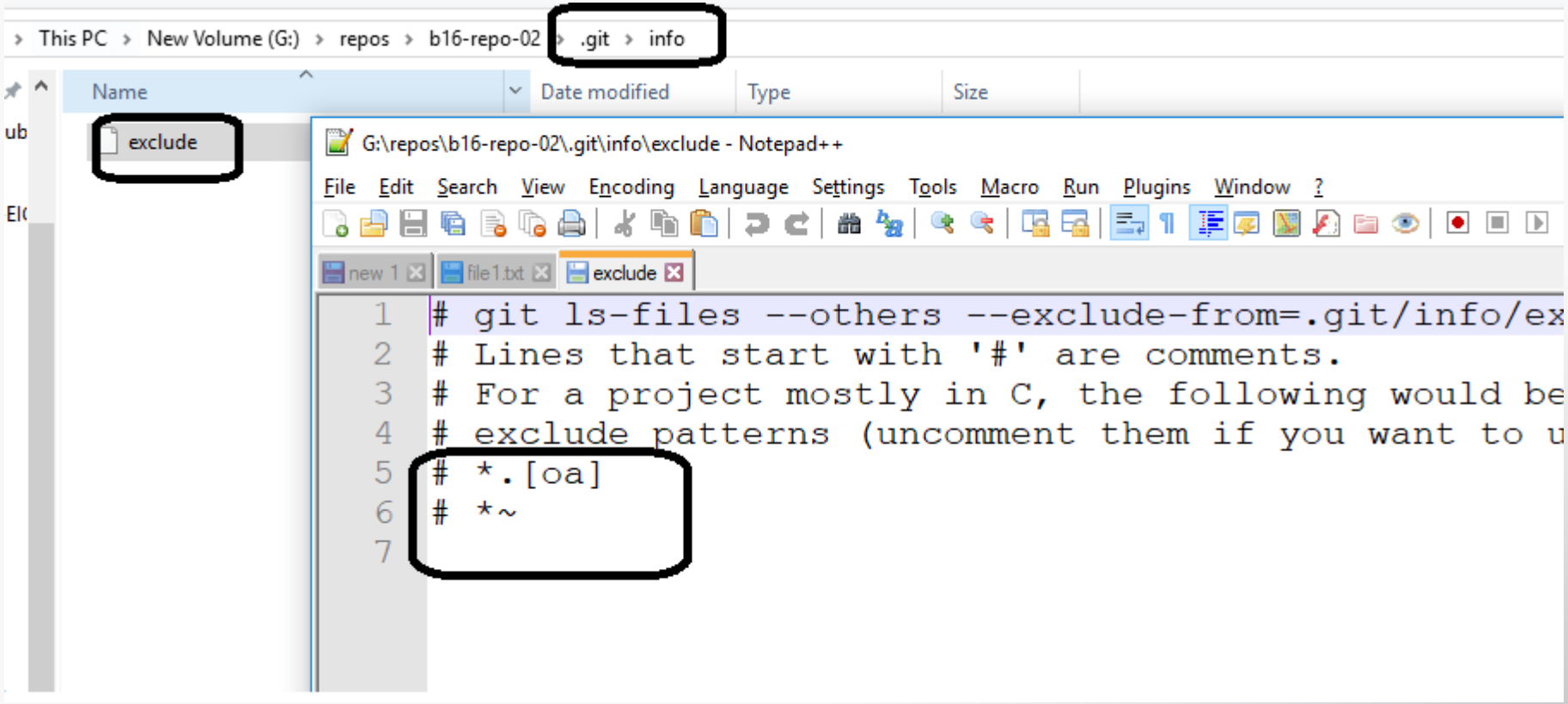
Git ignore patterns

Pattern	Example matches	Explanation*
**/logs	logs/debug.log logs/monday/foo.bar build/logs/debug.log	You can prepend a pattern with a double asterisk to match directories anywhere in the repository.
**/logs/debug.log	logs/debug.log build/logs/debug.log *but not* logs/build/debug.log	You can also use a double asterisk to match files based on their name and the name of their parent directory.
*.log	debug.log foo.log .log logs/debug.log	An asterisk is a wildcard that matches zero or more characters.
*.log !important.log	debug.log trace.log *but not* important.log logs/important.log	Prepending an exclamation mark to a pattern negates it. If a file matches a pattern, but *also* matches a negating pattern defined later in the file, it will not be ignored.
.log !important/.log trace.*	debug.log important/trace.log *but not* important/debug.log	Patterns defined after a negating pattern will re-ignore any previously negated files.

Git ignore patterns

Pattern	Example matches	Explanation*
/debug.log	debug.log *but not* logs/debug.log	Prepending a slash matches files only in the repository root.
debug.log	debug.log logs/debug.log	By default, patterns match files in any directory
debug?.log	debug0.log debugg.log *but not* debug10.log	A question mark matches exactly one character.
debug[0-9].log	debug0.log debug1.log *but not* debug10.log	Square brackets can also be used to match a single character from a specified range.
debug[01].log	debug0.log debug1.log *but not* debug2.log debug01.log	Square brackets match a single character form the specified set.

Personal Gitignore configuration file and location



Global Git ignore rules

- We can define global Git ignore patterns for all repositories on the local system by setting the git ***core.excludesFile*** property.

```
$ touch ~/.gitignore
$ git config --global core.excludesFile ~/.gitignore
```

Ignoring a previously committed file

- We can also un commit a file from the repo.
- Using the **–cached** option with **git rm**.

```
$ echo debug.log >> .gitignore
$ git rm --cached debug.log rm 'debug.log’
$ git commit -m "Start ignoring debug.log"
```

You can omit the **--cached** option if you want to delete the file from both the **repository and your local file system.**

To commit an ignored file

$ git add ***–f <filename>***

This would force commit the files.

GIT Stash

Git stash Working

- Stashes are encoded in your repository as commit objects.
- Special ref at **.git/refs/stash** points to your most recently created stash.
- Stash is just a commit, so we can inspect it with “git log”

Git stash

- ***Git stash*** temporarily stashes (Shelves) changes made to your working copy.
  - ➢This allows you to work on something else and re-apply them later.

```
$ git status
On branch master
Changes to be committed:
new file: style.css
Changes not staged for commit:
modified: index.html
$ git stash
Saved working directory and index state WIP on master:
HEAD is now at 5002d47 our new homepage
$ git status
On branch master
nothing to commit, working tree clean
```

Git stash Commands

- ***git stash pop*** -- Reapply previously stashed changes.
- ***git stash apply*** -- Reapply previously stashed changes and keeps the copy in the **stash area**.

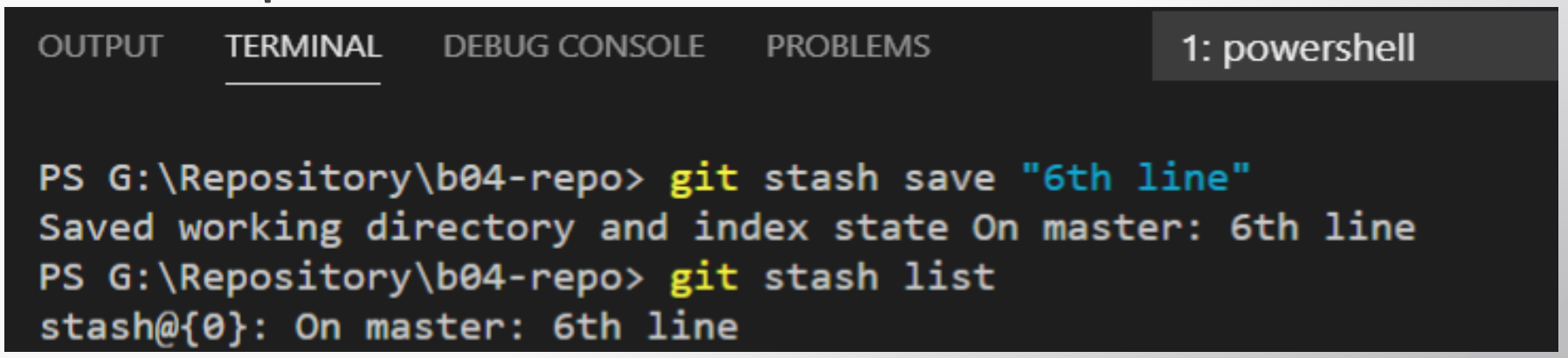
- **TIPS – BY Default File's that cannot be STASHED**
- New files not yet been staged.
- Also files that are ignored.
- ***git stash -u*** -- stash the untracked files.
- ***git stash -a*** -- stash the ignored files.

Managing Multiple stashes

- ***git stash list*** -- lists all the stashed

```
$ git stash list
stash@{0}: WIP on master: 5002d47 our new homepage
stash@{1}: WIP on master: 5002d47 our new homepage
stash@{2}: WIP on master: 5002d47 our new homepage
```

- ***git stash save "added 3rd line"*** – Good practice to add description.



Viewing stash diffs

- ***git stash show*** -- to View a summary of a stash

- ***git stash show -p*** -- To View the full diff of a stash

Cleaning up stash

- ***git stash drop 1*** -- this would drop the stash with index “1”.

- ***git stash clear*** -- To clear all the stashes saved

GIT Rebase -- Lets explore

Git rebase

- Rebasing is the process of moving or combining a sequence of commits to a new base commit.
- Rebasing is most useful and easily visualized in the context of a feature branching workflow.