ATLASSIAN Bitbucket **Tutorials** Enter Your Email For Git News

Beginner

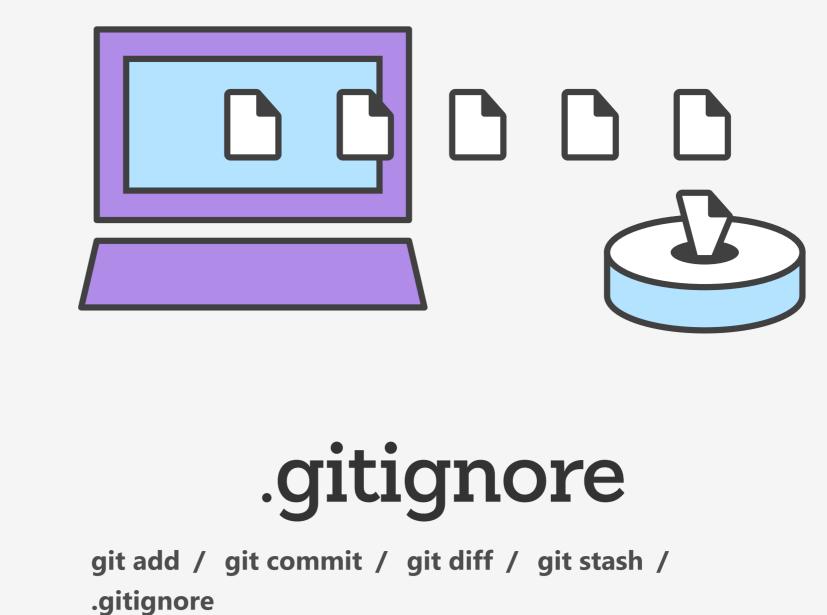
Learn Git

git add git commit git diff git stash .gitignore Inspecting a repository **Undoing changes**

Getting Started

Saving changes

Setting up a repository



Git sees every file in your working copy as one of three things: 1. tracked - a file which has been previously staged or committed;

2. untracked - a file which has not been staged or committed; or

- 3. ignored a file which Git has been explicitly told to ignore.
- Ignored files are usually build artifacts and machine generated files that can be derived from your repository source or should otherwise not be committed. Some common examples are:
 - dependency caches, such as the contents of /node_modules or /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
- 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 Shared .gitignore files in your repository Personal Git ignore rules Global Git ignore rules • Ignoring a previously committed file Committing an ignored file

Debugging .gitignore files

• Ignoring files in Git

- Stashing an ignored file
- Git ignore patterns
 - **Example matches**

logs/debug.log

logs/debug.log

but not

logs/monday/foo.bar

build/logs/debug.log

build/logs/debug.log

name of their parent logs/build/debug.log directory. debug.log An asterisk is a wildcard foo.log that matches zero or

Explanation*

You can prepend a

asterisk to match

the repository.

You can also use a

double asterisk to

match files based on

their name and the

pattern with a double

directories anywhere in

irectory of your re epository has mult epository root" wi	ns assume your .gitignore for pository, as is the conventicible .gitignore files, simply ith "directory containing the ing them, for the sanity of the sanit	on. If your mentally replace e .gitignore file"
logs/debug.log	logs/debug.log but not debug.log build/logs/debug.log	Patterns specifying a file in a particular directory are relative to the repository root. (You can prepend a slash if you like, but it doesn't do anything special.)
logs/*day/debug.log	logs/monday/debug.log logs/tuesday/debug.log but not logs/latest/debug.log	Wildcards can be used in directory names as well.
logs/**/debug.log	logs/debug.log logs/monday/debug.log logs/monday/pm/debug.log	A double asterisk matches zero or more directories.
logs/ !logs/important.log	logs/debug.log logs/important.log	Wait a minute! Shouldn't logs/important.log be negated in the example on the left Nope! Due to a performance-related quirk in Git, you can not negate a file that is ignored due to a pattern matching a directory
ogs/	logs/debug.log logs/latest/foo.bar build/logs/foo.bar build/logs/latest/debug.log	Appending a slash indicates the pattern is a directory. The entire contents of any directory in the repository matching that name – including all of its files and subdirectories – will be ignored
logs	logs logs/debug.log logs/latest/foo.bar build/logs build/logs/debug.log	If you don't append a slash, the pattern will match both files and the contents of directories with that name. In the example matches on the left, both directories and files named <i>logs</i> are ignored
debug[a-z].log	debuga.log debugb.log but not debug1.log	Ranges can be numeric or alphabetic.
debug[!01].log	debug2.log but not debug0.log debug1.log debug01.log	An exclamation mark can be used to match any character except one from the specified set.
debug[01].log	debug0.log debug1.log but not debug2.log debug01.log	Square brackets match a single character form the specified set.
debug[0-9].log	debug10.log debug1.log debug1.log but not debug10.log	Square brackets can also be used to match a single character from a specified range.
debug?. log	debug0.log debugg.log but not	A question mark matches exactly one character.
debug.log	logs/debug.log debug.log logs/debug.log	repository root. By default, patterns match files in any
trace.* /debug.log	important/debug.log debug.log but not	previously negated files. Prepending a slash matches files only in the
.log !important/.log	debug.log important/trace.log but not	defined later in the file, it will not be ignored. Patterns defined after a negating pattern will re-ignore any
t.log limportant.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 <i>also</i> matches a negating pattern

Shared .gitignore files in

Git ignore rules are usually defined in a .gitignore file at the

repository. Each pattern in a particular .gitignore file is tested

root of your repository. However, you can choose to define

multiple .gitignore files in different directories in your

it is versioned like any other file in your repository and shared with your teammates when you push. Typically you should only include patterns in .gitignore that will benefit other users of the repository.

You can also define personal ignore patterns for a particular

repository in a special file at .git/info/exclude. These are not

appropriate place to include patterns that will likely only benefit

versioned, and not distributed with your repository, so it's an

you. For example if you have a custom logging setup, or special development tools that produce files in your repository's working directory, you could consider adding them to .git/info/exclude to prevent them from being accidentally committed to your repository. Global Git ignore rules In addition, you can define global Git ignore patterns for all repositories on your local system by setting the Git

\$ git config --global core.excludesFile ~/.gitignore

candidates for ignoring globally.

You should be careful what patterns you choose to globally

temporary files created by some developer tools are typical

ignore, as different file types are relevant for different projects.

Special operating system files (e.g. .DS_Store and thumbs.db) or

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.

\$ cat .gitignore \$ git add -f debug.log \$ git commit -m "Force adding debug.log"

You might consider doing this if you have a general pattern (like

better solution is to define an exception to the general rule:

\$ echo !debug.log >> .gitignore

\$ git commit -m "Adding debug.log"

\$ cat .gitignore

\$ git add debug.log

ignored and untracked files as well.

!debug.log

teammates.

ignored:

*. log) defined, but you want to commit a specific file. However a

Stashing an ignored file

As you'd expect, by default git stash ignores ignored files and

can invoke git stash with the --all option to stash changes to

only stashes changes to files that are tracked by Git. However, you

This approach is more obvious, and less confusing, for your

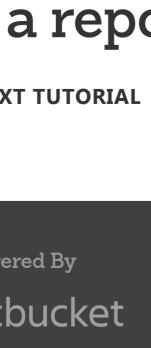
Debugging .gitignore files If you have complicated .gitignore patterns, or patterns spread

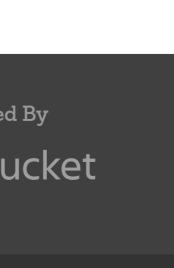
\$ git check-ignore -v debug.log .gitignore:3:*.log debug.log The output shows:

that exist in your repository.

Get started now

and the names themselves don't even have to correspond to files





Site hosted by **A** ATLASSIAN Enter Your Email For Git News

.gitignore uses globbing patterns to match against file names. You can construct your patterns using various symbols: **Pattern** **/logs **/logs/debug.log *. loa

comments in your .gitignore file:

You can use \ to escape .gitignore pattern characters if you have files or directories containing them: # ignore the file literally named foo[01].txt your repository

relative to the directory containing that file. However the convention, and simplest approach, is to define a single .gitignore file in the root. As your .gitignore file is checked in, Personal Git ignore rules

core. excludesFile property. You'll have to create this file yourself. If you're unsure where to put your global .gitignore file, your home directory isn't a bad choice (and makes it easy to find later). Once you've created the file, you'll need to configure its location with git config:

\$ touch ~/.gitignore

Ignoring a previously committed file If you want to ignore a file that you've committed in the past, you'll need to delete the file from your repository and then add a .gitignore rule for it. Using the --cached option with git rm

means that the file will be deleted from your repository, but will

remain in your working directory as an ignored file.

\$ echo debug.log >> .gitignore

\$ git rm --cached debug.log

file It is possible to force an ignored file to be committed to the repository using the -f (or --force) option with git add:

Committing an ignored

git stash is a powerful Git feature for temporarily shelving and reverting local changes, allowing you to re-apply them later on.

over multiple .gitignore files, it can be difficult to track down why a particular file is being ignored. You can use the git check-ignore command with the -v (or --verbose) option to determine which pattern is causing a particular file to be

<file containing the pattern> : e number of the pattern> : You can pass multiple file names to git check-ignore if you like,

Ready to learn Git? Try this interactive tutorial.

Next up: Inspecting a repository **START NEXT TUTORIAL**

License.

Powered By **■** Bitbucket Recommend Except where otherwise noted, all content is licensed under a Creative Commons Attribution 2.5 Australia

Want future articles?