Software Development

Setting up a repository

git init

git clone

git config

git alias

Saving changes (Git add)

git diff

git stash

.gitignore

Inspecting a repository

git tag

Undoing changes

git blame

git clean

git revert

git reset

git rebase

git reflog

git rm

Rewriting history

> Collaborating workflows

git commit

Git ignore > Learn Git > Beginner

✓ Getting started

DevOps ∨ Agile

Git sees every file in your working copy as one of three things:

examples are:

Microservices

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

• dependency caches, such as the contents of /node_modules or /packages

from your repository source or should otherwise not be committed. Some common

• 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

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. In this document, we'll cover:

your repository. There is no explicit git ignore command: instead the .gitignore file must be

• <u>Shared .gitignore files in your repository</u>

Git ignore patterns

• Personal Git ignore rules Advanced Tips

• Global Git ignore rules Articles

• <u>Ignoring a previously committed file</u> • Committing an ignored file

• <u>Debugging .gitignore files</u>

Git ignore patterns

• Stashing an ignored file

Pattern	Example matches	Explanation*
**/logs	logs/debug.log	You can prepend a pattern
	logs/monday/foo.bar	with a double asterisk to
	build/logs/debug.log	match directories anywhere i the repository.
**/logs/debug.log	logs/debug.log	
	build/logs/debug.log	You can also use a double asterisk to match files based
	but not	on their name and the name
	logs/build/debug.log	their parent directory.
*.log	debug.log	
	foo.log	An asterisk is a wildcard tha
	.log	matches zero or more
	logs/debug.log	characters.
		Duran and in a sure and a sure at it as
	debug.log	Prepending an exclamation mark to a pattern negates it.
*.log	but not	a file matches a pattern, bu
!important.log	logs/debug.log	also matches a negating pattern defined later in the fi
		it will not be ignored.
/debug.log	debug.log	Patterns defined after a
	but not	negating pattern will re-igno
	logs/debug.log	any previously negated files
dobug la-	debug.log	Prepending a slash matches
debug.log	logs/debug.log	files only in the repository roo
debug?. <mark>log</mark>	debug0.log	
	debugg.log	
	but not	A question mark matches exactly one character.
	debug10.log	exactly one character.
debug[0-9]. <mark>log</mark>	debug0.log	Course horselests and also he
	debug1.log	Square brackets can also be used to match a single
	but not	character from a specified
	debug10.log	range.
debug[01]. <mark>log</mark>		
	debug0.log	
	debug1.log but not	Square brackets match a single character form the
	debug2.log	specified set.
	debug01.log	
	debug2.log but not	An exclamation mark can be
debug[!01].log	debug0.log	used to match any characte
	debug1.log	except one from the specifie set.
	debug01.log	
	debuga.log	
		Panges can be numeric or
debug[a-z].log	debugb.log but not	Ranges can be numeric or alphabetic.
	debug1.log	
logs	logs	If you don't append a slash, t
	logs/debug.log	pattern will match both file
	logs/latest/foo.bar	and the contents of directoric with that name. In the
	build/logs	example matches on the lef
	build/logs/debug.log	both directories and files named <i>logs</i> are ignored
logs/		Appending a slash indicates
	logs/debug.log	the pattern is a directory. Th
	logs/latest/foo.bar	in the repository matching th
	build/logs/foo.bar	name – including all of its file and subdirectories – will be
	build/logs/latest/debug.log	ignored
		Wait a minute! Shouldn't
		logs/important.log be
		negated in the example on th

logs/debug.log

logs/important.log

logs/debug.log

logs/monday/debug.log

logs/monday/pm/debug.log

logs/monday/debug.log

logs/tuesday/debug.log

but not

logs/latest/debug.log

logs/debug.log

but not

debug.log

build/logs/debug.log

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

** these explanations assume your .gitignore file is in the top level

multiple .gitignore files, simply mentally replace "repository root"

with "directory containing the .gitignore file" (and consider unifying

In addition to these characters, you can use # to include comments in

directory of your repository, as is the convention. If your repository has

logs/

!logs/important.log

logs/**/debug.log

logs/*day/debug.log

logs/debug.log

them, for the sanity of your team).*

ignore all logs

your .gitignore file:

Read tutorial → Shared .gitignore files in your repository Git ignore rules are usually defined in a .gitignore file at the root of your repository. However, you can

negated in the example on the left

Nope! Due to a performancerelated quirk in Git, you can not negate a file that is ignored due to a pattern matching a directory

A double asterisk matches

zero or more directories.

Wildcards can be used in

directory names as well.

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.)

6

RELATED MATERIAL

Git branch

Read article →

SEE SOLUTION

Learn Git with **Bitbucket Cloud**

repository. Personal Git ignore rules

choose to define multiple .gitignore files in different directories in your repository. Each pattern in a

and simplest approach, is to define a single .gitignore file in the root. As your .gitignore file is

push. Typically you should only include patterns in .gitignore that will benefit other users of the

particular .gitignore file is tested relative to the directory containing that file. However the convention,

checked in, it is versioned like any other file in your repository and shared with your teammates when you

You can also define personal ignore patterns for a particular repository in a special file at .git/info/exclude . These are not versioned, and not distributed with your repository, so it's an appropriate place to include patterns that will likely only benefit 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

your global .gitignore file, your home directory isn't a bad choice (and makes it easy to find later). Once

Git Core.excludesFile property. You'll have to create this file yourself. If you're unsure where to put

you've created the file, you'll need to configure its location with <code>git config</code>:

created by some developer tools are typical candidates for ignoring globally.

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

You should be careful what patterns you choose to globally ignore, as different file types are relevant for

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

\$ echo debug.log >> .gitignore \$ git rm --cached debug.log rm 'debug.log'

\$ git commit -m "Start ignoring debug.log"

Committing an ignored file

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

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

\$ git commit -m "Force adding debug.log" You might consider doing this if you have a general pattern (like *.log) defined, but you want to commit

\$ echo !debug.log >> .gitignore \$ cat .gitignore *.log

\$ git commit -m "Adding debug.log" This approach is more obvious, and less confusing, for your teammates. Stashing an ignored file

re-apply them later on. As you'd expect, by default <code>git stash</code> ignores ignored files and only stashes changes to files that are tracked by Git. However, you can invoke git stash with the --all option to stash

Debugging .gitignore files If you have complicated .gitignore patterns, or patterns spread over multiple .gitignore files, it can

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

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

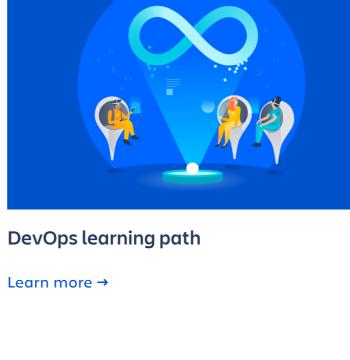
<file containing the pattern> : e number of the pattern> : <pattern></par>

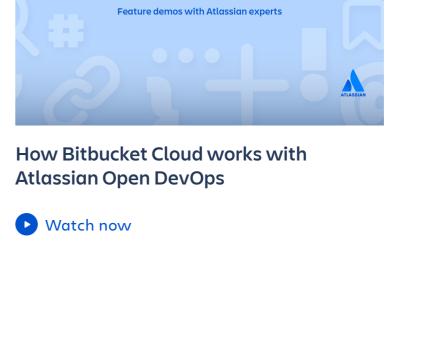
have to correspond to files that exist in your repository. SHARE THIS ARTICLE **NEXT TOPIC**

Recommended reading Bookmark these resources to learn about types of DevOps teams, or for ongoing updates about DevOps at Atlassian.

f y in







Demo Den

<fil

Inspecting a

repository →

Sign up

PRODUCTS

Technical Support

Purchasing & licensing

Events Blogs Atlassian Foundation **Investor Relations** Contact us

\$ cat .gitignore *.log \$ git add -f debug.log

a specific file. However a better solution is to define an exception to the general rule: !debug.log

with git add:

git stash is a powerful Git feature for temporarily shelving and reverting local changes, allowing you to

changes to ignored and untracked files as well.

\$ git add debug.log

The output shows:

You can pass multiple file names to git check-ignore if you like, and the names themselves don't even

Sign up for our DevOps newsletter

ABOUT ATLASSIAN

Company

Careers

⊕ English ▼ Privacy policy

Email address

A ATLASSIAN

View all products Notice at Collection

Copyright © 2024 Atlassian Impressum

View all resources

Trust & Security