

Git®

Notes for Professionals

Chapter 2: Browsing the history

Parameter	Explanation
-q, --quiet	Quiet, suppresses diff output
--source	Shows source of commit
--use-mailmap	Use mail map file (changes user info for committing user)
--decorate[<mode>]	Decorate options
-L, --limit[<n>]	Show log for specific range of lines in the source (from 1. Starts to n. Also shows diff)
--show-signature	Display signatures of signed commits
-i, --regexp-ignore-case	Match the regular expression limiting patterns without regard to case

Section 2.1: "Regular" Git Log

`git log` will display all your commits with the author and hash. This will be shown over multiple lines to show a single line per commit, look at swimming). Use the `q` key to exit the log.

By default, with no arguments, `git log` lists the commits made in that repository in order – that is, the most recent commits show up first. As you can see, the commit with its SHA-1 checksum, the author's name and email, the date written, and the source.

Example (from [FreeCodeCamp](#) repository)

```
commit 974f97590242748c425982776f1a05780000cf
Merge: 6b1f94 e8db726
Author: Thu Mar 24 15:52:07 2016 -0700
Date: Thu Mar 24 15:52:07 2016 -0700
Merge: pull request #7724 from BKLinahan/fix-where-art-thou
File: 'its' type in Where Art Thou description
commit 6b1f94e8db72656e080e4e0d797c796f65af270e5
Author: BKLinahan
Date: Thu Mar 24 21:13:30 2016 -0800
File: 'its' type in Where Art Thou description
commit a034f0e2493054155c0435f3270c1025598e7
Merge: 1001870 2653d01
Author: Reugun Nonawatra
Date: Thu Mar 24 14:26:04 2016 -0530
Merge: pull request #7719 from deatheyko477/fix/unnecessary-commits
Reverse unnecessary commits from COMMITBUT1300, add
```

If you wish to limit your command to last `n` commits log you can simply pass a parameter. For example, if you wish to list last 2 commits log:

Chapter 10: Committing

Parameter	Details
--message -m	Message to include in the commit. Specifying this parameter bypasses Git's normal behavior of opening an editor.
--amend	Specify that the changes currently staged should be added (amended) to the previous commit. Be careful, this can rewrite history!
--no-edit	Use the selected commit message without launching an editor. For example, <code>git commit --amend --no-edit</code> amends a commit without changing its commit message.
--no-e	Commit all changes, including changes that aren't yet staged.
--date	Manually set the date that will be associated with the commit.
--only	Commit only the paths specified. This will not commit what you currently have staged unless told to do so.
--patch -p	Use the interactive patch selection interface to choose which changes to commit.
--help	Displays the man page for <code>git commit</code> .
-S, --sign	Sign commit, GPG-sign commit, countermand <code>commit.gpgsign</code> configuration variable.
-i, --no-gpg-sign	This option bypasses the pre-commit and commit-msg hooks. See also hooks

Commits with Git provide accountability by attributing authors with changes to code. Git offers multiple features for the specificity and security of commits. This topic explains and demonstrates proper practices and procedures in committing with Git.

Section 10.1: Stage and commit changes

The basics
After making changes to your source code, you should **stage** these changes with Git before you can commit them.

For example if you change `README.md` and `program.py`:

```
git add README.md program.py
```

This tells git that you want to add the files to the next commit you do.

Then, commit your changes with:

```
git commit
```

Note that this will open a text editor, which is often vim. If you are not familiar with vim, you might want to use that you can press `!` to go into insert mode, write your commit message, then press `Esc` and `:wq` to save and avoid opening the text editor, simply include the `-m` flag with your message.

```
git commit -m "Commit message here"
```

Commit messages often follow some specific formatting rules. See Good commit messages for more info.

Shortcuts

If you have changed a lot of files in the directory, rather than listing each one of them, you could use:

```
git commit -a
```

Chapter 25: Cloning Repositories

Section 25.1: Shallow Clone

Cloning a huge repository (like a project with multiple years of history) might take a long time, or fail because of the amount of data to be transferred. In cases where you don't need to have the full history available, you can do a shallow clone:

```
git clone [repo, url] --depth 1
```

The above command will fetch just the last commit (from the remote repository).

Be aware that you may not be able to resolve merges in a shallow repository. It's often a good idea to take at least as many commits as you are going to need to backtrack to resolve merges. For example, to instead get the last 50 commits:

```
git clone [repo, url] --depth 50
```

Later, if required, you can fetch the rest of the repository:

```
version = 1.0.3
git fetch --unshallow # equivalent of git fetch --depth=174405047
version = 1.0.3
git fetch --depth=1000 # fetch the last 1000 commits
```

Section 25.2: Regular Clone

To download the entire repository including the full history and all branches, type:

```
git clone url
```

The example above will place it in a directory with the same name as the repository name.

To download the repository and save it in a specific directory, type:

```
git clone url <directory>
```

For more details, visit [Clone a repository](#).

Section 25.3: Clone a specific branch

To clone a specific branch of a repository, type `--branch <branch name>` before the repository url:

```
git clone --branch <branch name> url <directory>
```

To use the shorthand option for `--branch`, type `-b`. This command downloads entire repository and checks out `<branch name>`.

To save disk space you can clone history loading only to single branch with:

```
git clone --branch <branch name> --single-branch url <directory>
```

Contents

About	1
Chapter 1: Getting started with Git	2
Section 1.1: Create your first repository, then add and commit files	2
Section 1.2: Clone a repository	4
Section 1.3: Sharing code	4
Section 1.4: Setting your user name and email	5
Section 1.5: Setting up the upstream remote	6
Section 1.6: Learning about a command	6
Section 1.7: Set up SSH for Git	6
Section 1.8: Git Installation	7
Chapter 2: Browsing the history	10
Section 2.1: "Regular" Git Log	10
Section 2.2: Prettier log	11
Section 2.3: Colorize Logs	11
Section 2.4: Oneline log	11
Section 2.5: Log search	12
Section 2.6: List all contributions grouped by author name	12
Section 2.7: Searching commit string in git log	13
Section 2.8: Log for a range of lines within a file	14
Section 2.9: Filter logs	14
Section 2.10: Log with changes inline	14
Section 2.11: Log showing committed files	15
Section 2.12: Show the contents of a single commit	15
Section 2.13: Git Log Between Two Branches	16
Section 2.14: One line showing commiter name and time since commit	16
Chapter 3: Working with Remotes	17
Section 3.1: Deleting a Remote Branch	17
Section 3.2: Changing Git Remote URL	17
Section 3.3: List Existing Remotes	17
Section 3.4: Removing Local Copies of Deleted Remote Branches	17
Section 3.5: Updating from Upstream Repository	18
Section 3.6: ls-remote	18
Section 3.7: Adding a New Remote Repository	18
Section 3.8: Set Upstream on a New Branch	18
Section 3.9: Getting Started	19
Section 3.10: Renaming a Remote	19
Section 3.11: Show information about a Specific Remote	20
Section 3.12: Set the URL for a Specific Remote	20
Section 3.13: Get the URL for a Specific Remote	20
Section 3.14: Changing a Remote Repository	20
Chapter 4: Staging	21
Section 4.1: Staging All Changes to Files	21
Section 4.2: Unstage a file that contains changes	21
Section 4.3: Add changes by hunk	21
Section 4.4: Interactive add	22
Section 4.5: Show Staged Changes	22
Section 4.6: Staging A Single File	23

Section 4.7: Stage deleted files	23
Chapter 5: Ignoring Files and Folders	24
Section 5.1: Ignoring files and directories with a .gitignore file	24
Section 5.2: Checking if a file is ignored	26
Section 5.3: Exceptions in a .gitignore file	27
Section 5.4: A global .gitignore file	27
Section 5.5: Ignore files that have already been committed to a Git repository	27
Section 5.6: Ignore files locally without committing ignore rules	28
Section 5.7: Ignoring subsequent changes to a file (without removing it)	29
Section 5.8: Ignoring a file in any directory	29
Section 5.9: Prefilled .gitignore Templates	29
Section 5.10: Ignoring files in subfolders (Multiple gitignore files)	30
Section 5.11: Create an Empty Folder	31
Section 5.12: Finding files ignored by .gitignore	31
Section 5.13: Ignoring only part of a file [stub]	32
Section 5.14: Ignoring changes in tracked files. [stub]	33
Section 5.15: Clear already committed files, but included in .gitignore	34
Chapter 6: Git Diff	35
Section 6.1: Show differences in working branch	35
Section 6.2: Show changes between two commits	35
Section 6.3: Show differences for staged files	35
Section 6.4: Comparing branches	36
Section 6.5: Show both staged and unstaged changes	36
Section 6.6: Show differences for a specific file or directory	36
Section 6.7: Viewing a word-diff for long lines	37
Section 6.8: Show differences between current version and last version	37
Section 6.9: Produce a patch-compatible diff	37
Section 6.10: difference between two commit or branch	38
Section 6.11: Using meld to see all modifications in the working directory	38
Section 6.12: Diff UTF-16 encoded text and binary plist files	38
Chapter 7: Undoing	40
Section 7.1: Return to a previous commit	40
Section 7.2: Undoing changes	40
Section 7.3: Using reflog	41
Section 7.4: Undoing merges	41
Section 7.5: Revert some existing commits	43
Section 7.6: Undo / Redo a series of commits	43
Chapter 8: Merging	45
Section 8.1: Automatic Merging	45
Section 8.2: Finding all branches with no merged changes	45
Section 8.3: Aborting a merge	45
Section 8.4: Merge with a commit	45
Section 8.5: Keep changes from only one side of a merge	45
Section 8.6: Merge one branch into another	46
Chapter 9: Submodules	47
Section 9.1: Cloning a Git repository having submodules	47
Section 9.2: Updating a Submodule	47
Section 9.3: Adding a submodule	47
Section 9.4: Setting a submodule to follow a branch	48
Section 9.5: Moving a submodule	48

Section 9.6: Removing a submodule	49
Chapter 10: Committing	50
Section 10.1: Stage and commit changes	50
Section 10.2: Good commit messages	51
Section 10.3: Amending a commit	52
Section 10.4: Committing without opening an editor	53
Section 10.5: Committing changes directly	53
Section 10.6: Selecting which lines should be staged for committing	53
Section 10.7: Creating an empty commit	54
Section 10.8: Committing on behalf of someone else	54
Section 10.9: GPG signing commits	55
Section 10.10: Committing changes in specific files	55
Section 10.11: Committing at a specific date	55
Section 10.12: Amending the time of a commit	56
Section 10.13: Amending the author of a commit	56
Chapter 11: Aliases	57
Section 11.1: Simple aliases	57
Section 11.2: List / search existing aliases	57
Section 11.3: Advanced Aliases	57
Section 11.4: Temporarily ignore tracked files	58
Section 11.5: Show pretty log with branch graph	58
Section 11.6: See which files are being ignored by your .gitignore configuration	59
Section 11.7: Updating code while keeping a linear history	60
Section 11.8: Unstage staged files	60
Chapter 12: Rebasing	61
Section 12.1: Local Branch Rebasing	61
Section 12.2: Rebase: ours and theirs, local and remote	61
Section 12.3: Interactive Rebase	63
Section 12.4: Rebase down to the initial commit	64
Section 12.5: Configuring autostash	64
Section 12.6: Testing all commits during rebase	65
Section 12.7: Rebasing before a code review	65
Section 12.8: Aborting an Interactive Rebase	67
Section 12.9: Setup git-pull for automatically perform a rebase instead of a merge	68
Section 12.10: Pushing after a rebase	68
Chapter 13: Configuration	69
Section 13.1: Setting which editor to use	69
Section 13.2: Auto correct typos	69
Section 13.3: List and edit the current configuration	70
Section 13.4: Username and email address	70
Section 13.5: Multiple usernames and email address	70
Section 13.6: Multiple git configurations	71
Section 13.7: Configuring line endings	72
Section 13.8: configuration for one command only	72
Section 13.9: Setup a proxy	72
Chapter 14: Branching	74
Section 14.1: Creating and checking out new branches	74
Section 14.2: Listing branches	75
Section 14.3: Delete a remote branch	75
Section 14.4: Quick switch to the previous branch	76

Section 14.5: Check out a new branch tracking a remote branch	76
Section 14.6: Delete a branch locally	76
Section 14.7: Create an orphan branch (i.e. branch with no parent commit)	77
Section 14.8: Rename a branch	77
Section 14.9: Searching in branches	77
Section 14.10: Push branch to remote	77
Section 14.11: Move current branch HEAD to an arbitrary commit	78
Chapter 15: Rev-List	79
Section 15.1: List Commits in master but not in origin/master	79
Chapter 16: Squashing	80
Section 16.1: Squash Recent Commits Without Rebasing	80
Section 16.2: Squashing Commit During Merge	80
Section 16.3: Squashing Commits During a Rebase	80
Section 16.4: Autosquashing and fixups	81
Section 16.5: Autosquash: Committing code you want to squash during a rebase	82
Chapter 17: Cherry Picking	83
Section 17.1: Copying a commit from one branch to another	83
Section 17.2: Copying a range of commits from one branch to another	83
Section 17.3: Checking if a cherry-pick is required	84
Section 17.4: Find commits yet to be applied to upstream	84
Chapter 18: Recovering	85
Section 18.1: Recovering from a reset	85
Section 18.2: Recover from git stash	85
Section 18.3: Recovering from a lost commit	86
Section 18.4: Restore a deleted file after a commit	86
Section 18.5: Restore file to a previous version	86
Section 18.6: Recover a deleted branch	87
Chapter 19: Git Clean	88
Section 19.1: Clean Interactively	88
Section 19.2: Forcefully remove untracked files	88
Section 19.3: Clean Ignored Files	88
Section 19.4: Clean All Untracked Directories	88
Chapter 20: Using a .gitattributes file	90
Section 20.1: Automatic Line Ending Normalization	90
Section 20.2: Identify Binary Files	90
Section 20.3: Prefilled .gitattribute Templates	90
Section 20.4: Disable Line Ending Normalization	90
Chapter 21: .mailmap file: Associating contributor and email aliases	91
Section 21.1: Merge contributors by aliases to show commit count in shortlog	91
Chapter 22: Analyzing types of workflows	92
Section 22.1: Centralized Workflow	92
Section 22.2: Gitflow Workflow	93
Section 22.3: Feature Branch Workflow	95
Section 22.4: GitHub Flow	95
Section 22.5: Forking Workflow	96
Chapter 23: Pulling	97
Section 23.1: Pulling changes to a local repository	97
Section 23.2: Updating with local changes	98
Section 23.3: Pull, overwrite local	98

[Click here to download full PDF material](#)