

# 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 for changes (user info for committing user).
--decorate= <i>c</i>	Decorate options: -r, --raw: raw -s, --short: short -t, --oneline: oneline -m, --no-prefix: no prefix -L, --list-style: list style -c, --color: color -4, --regen-ignores-case: Match the regular expression limiting patterns without regard to case.
--show-signature	Display signatures of signed commits.
-4, --regen-ignores-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 pages. If you want to see the commits in a single line per commit, look at [stitching](#). Use the `q` key to exit the tool.

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

Example (from [freeCodeCamp](#) repository):

```
commit 39e792759e22140c47508276f1440748000bcf  
Author: Brian  
Date: Thu Mar 24 15:52:07 2016 -0700  
Merge pull request #7726 from BKLinhbantu/where-art-there  
Fix 'its' typo in Where Art Thou description  
commit e6b02700b16eabef0e6797c796fbaf270e5  
Author: BKLinhb  
Date: Thu Mar 24 21:13:36 2016 -0700  
Fix 'its' typo in Where Art Thou description  
commit e6b02700b16eabef0e6797c796fbaf270e5  
Author: BKLinhb  
Date: Thu Mar 24 21:13:36 2016 -0700  
Merge pull request #7719 from danthonyho42/fix/unnecessary-  
Review unnecessary .cover from CONTRIBUTING.md  
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 --no-edit</code> amends a commit without changing its commit message.
--all, -a	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 main page for <code>git commit</code> .
--signoff, -S --gpg-sign, -s --gpg-sign	Start commit, GPG-sign commit, countermand <code>git gpg-sign</code> configuration variable.
--no-verify	This option bypasses the pre-commit and committing hooks. See also <a href="#">Hooks</a> .

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** those 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 know that you can press `i` to go into **insert mode**, write your commit message, then press `Esc` and `:wq` to save and exit 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 information.

##### Shortcuts

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

[Git Notes for Professionals](#)

### 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, or fail because of the 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 one commit.

```
git clone [repo_url] --depth 50
```

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

```
git fetch --unshallow  
# equivalent of git fetch --depth=2147483647  
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.

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 `-b branch` `--branch branch_name` before the repository url:

```
git clone --branch <branch_name> <url> [directory]
```

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

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

```
git clone --branch <branch_name> --single-branch <url> [directory]
```

[Git Notes for Professionals](#)

**100+ pages**  
of professional hints and tricks

# Contents

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

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

Section 9.6: Removing a submodule .....	49
<b>Chapter 10: Committing .....</b>	<b>50</b>
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
<b>Chapter 11: Aliases .....</b>	<b>57</b>
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
<b>Chapter 12: Rebasing .....</b>	<b>61</b>
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
<b>Chapter 13: Configuration .....</b>	<b>69</b>
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
<b>Chapter 14: Branching .....</b>	<b>74</b>
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
<b>Chapter 15: Rev-List</b> .....	79
Section 15.1: List Commits in master but not in origin/master .....	79
<b>Chapter 16: Squashing</b> .....	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
<b>Chapter 17: Cherry Picking</b> .....	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
<b>Chapter 18: Recovering</b> .....	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
<b>Chapter 19: Git Clean</b> .....	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
<b>Chapter 20: Using a .gitattributes file</b> .....	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
<b>Chapter 21: .mailmap file: Associating contributor and email aliases</b> .....	91
Section 21.1: Merge contributors by aliases to show commit count in shortlog .....	91
<b>Chapter 22: Analyzing types of workflows</b> .....	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
<b>Chapter 23: Pulling</b> .....	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

<a href="#">Section 23.4: Pull code from remote</a>	98
<a href="#">Section 23.5: Keeping linear history when pulling</a>	98
<a href="#">Section 23.6: Pull, "permission denied"</a>	99
<b><a href="#">Chapter 24: Hooks</a></b>	100
<a href="#">Section 24.1: Pre-push</a>	100
<a href="#">Section 24.2: Verify Maven build (or other build system) before committing</a>	101
<a href="#">Section 24.3: Automatically forward certain pushes to other repositories</a>	101
<a href="#">Section 24.4: Commit-msg</a>	102
<a href="#">Section 24.5: Local hooks</a>	102
<a href="#">Section 24.6: Post-checkout</a>	102
<a href="#">Section 24.7: Post-commit</a>	103
<a href="#">Section 24.8: Post-receive</a>	103
<a href="#">Section 24.9: Pre-commit</a>	103
<a href="#">Section 24.10: Prepare-commit-msg</a>	103
<a href="#">Section 24.11: Pre-rebase</a>	103
<a href="#">Section 24.12: Pre-receive</a>	104
<a href="#">Section 24.13: Update</a>	104
<b><a href="#">Chapter 25: Cloning Repositories</a></b>	105
<a href="#">Section 25.1: Shallow Clone</a>	105
<a href="#">Section 25.2: Regular Clone</a>	105
<a href="#">Section 25.3: Clone a specific branch</a>	105
<a href="#">Section 25.4: Clone recursively</a>	106
<a href="#">Section 25.5: Clone using a proxy</a>	106
<b><a href="#">Chapter 26: Stashing</a></b>	107
<a href="#">Section 26.1: What is Stashing?</a>	107
<a href="#">Section 26.2: Create stash</a>	108
<a href="#">Section 26.3: Apply and remove stash</a>	109
<a href="#">Section 26.4: Apply stash without removing it</a>	109
<a href="#">Section 26.5: Show stash</a>	109
<a href="#">Section 26.6: Partial stash</a>	109
<a href="#">Section 26.7: List saved stashes</a>	110
<a href="#">Section 26.8: Move your work in progress to another branch</a>	110
<a href="#">Section 26.9: Remove stash</a>	110
<a href="#">Section 26.10: Apply part of a stash with checkout</a>	110
<a href="#">Section 26.11: Recovering earlier changes from stash</a>	110
<a href="#">Section 26.12: Interactive Stashing</a>	111
<a href="#">Section 26.13: Recover a dropped stash</a>	111
<b><a href="#">Chapter 27: Subtrees</a></b>	113
<a href="#">Section 27.1: Create, Pull, and Backport Subtree</a>	113
<b><a href="#">Chapter 28: Renaming</a></b>	114
<a href="#">Section 28.1: Rename Folders</a>	114
<a href="#">Section 28.2: rename a local and the remote branch</a>	114
<a href="#">Section 28.3: Renaming a local branch</a>	114
<b><a href="#">Chapter 29: Pushing</a></b>	115
<a href="#">Section 29.1: Push a specific object to a remote branch</a>	115
<a href="#">Section 29.2: Push</a>	116
<a href="#">Section 29.3: Force Pushing</a>	117
<a href="#">Section 29.4: Push tags</a>	117
<a href="#">Section 29.5: Changing the default push behavior</a>	117

<b>Chapter 30: Internals</b> .....	119
Section 30.1: Repo .....	119
Section 30.2: Objects .....	119
Section 30.3: HEAD ref .....	119
Section 30.4: Refs .....	119
Section 30.5: Commit Object .....	120
Section 30.6: Tree Object .....	121
Section 30.7: Blob Object .....	121
Section 30.8: Creating new Commits .....	122
Section 30.9: Moving HEAD .....	122
Section 30.10: Moving refs around .....	122
Section 30.11: Creating new Refs .....	122
<b>Chapter 31: git-tfs</b> .....	123
Section 31.1: git-tfs clone .....	123
Section 31.2: git-tfs clone from bare git repository .....	123
Section 31.3: git-tfs install via Chocolatey .....	123
Section 31.4: git-tfs Check In .....	123
Section 31.5: git-tfs push .....	123
<b>Chapter 32: Empty directories in Git</b> .....	124
Section 32.1: Git doesn't track directories .....	124
<b>Chapter 33: git-svn</b> .....	125
Section 33.1: Cloning the SVN repository .....	125
Section 33.2: Pushing local changes to SVN .....	125
Section 33.3: Working locally .....	125
Section 33.4: Getting the latest changes from SVN .....	126
Section 33.5: Handling empty folders .....	126
<b>Chapter 34: Archive</b> .....	127
Section 34.1: Create an archive of git repository .....	127
Section 34.2: Create an archive of git repository with directory prefix .....	127
Section 34.3: Create archive of git repository based on specific branch, revision, tag or directory .....	128
<b>Chapter 35: Rewriting history with filter-branch</b> .....	129
Section 35.1: Changing the author of commits .....	129
Section 35.2: Setting git committer equal to commit author .....	129
<b>Chapter 36: Migrating to Git</b> .....	130
Section 36.1: SubGit .....	130
Section 36.2: Migrate from SVN to Git using Atlassian conversion utility .....	130
Section 36.3: Migrating Mercurial to Git .....	131
Section 36.4: Migrate from Team Foundation Version Control (TFVC) to Git .....	131
Section 36.5: Migrate from SVN to Git using svn2git .....	132
<b>Chapter 37: Show</b> .....	133
Section 37.1: Overview .....	133
<b>Chapter 38: Resolving merge conflicts</b> .....	134
Section 38.1: Manual Resolution .....	134
<b>Chapter 39: Bundles</b> .....	135
Section 39.1: Creating a git bundle on the local machine and using it on another .....	135
<b>Chapter 40: Display commit history graphically with Gitk</b> .....	136
Section 40.1: Display commit history for one file .....	136
Section 40.2: Display all commits between two commits .....	136
Section 40.3: Display commits since version tag .....	136

<b>Chapter 41: Bisecting/Finding faulty commits</b> .....	137
Section 41.1: Binary search (git bisect) .....	137
Section 41.2: Semi-automatically find a faulty commit .....	137
<b>Chapter 42: Blaming</b> .....	139
Section 42.1: Only show certain lines .....	139
Section 42.2: To find out who changed a file .....	139
Section 42.3: Show the commit that last modified a line .....	140
Section 42.4: Ignore whitespace-only changes .....	140
<b>Chapter 43: Git revisions syntax</b> .....	141
Section 43.1: Specifying revision by object name .....	141
Section 43.2: Symbolic ref names: branches, tags, remote-tracking branches .....	141
Section 43.3: The default revision: HEAD .....	141
Section 43.4: Reflog references: <refname>@{<n>} .....	141
Section 43.5: Reflog references: <refname>@{<date>} .....	142
Section 43.6: Tracked / upstream branch: <branchname>@{upstream} .....	142
Section 43.7: Commit ancestry chain: <rev>^, <rev>~<n>, etc .....	142
Section 43.8: Dereferencing branches and tags: <rev>^0, <rev>^<type> .....	143
Section 43.9: Youngest matching commit: <rev>^{/<text>}, :/<text> .....	143
<b>Chapter 44: Worktrees</b> .....	145
Section 44.1: Using a worktree .....	145
Section 44.2: Moving a worktree .....	145
<b>Chapter 45: Git Remote</b> .....	147
Section 45.1: Display Remote Repositories .....	147
Section 45.2: Change remote url of your Git repository .....	147
Section 45.3: Remove a Remote Repository .....	148
Section 45.4: Add a Remote Repository .....	148
Section 45.5: Show more information about remote repository .....	148
Section 45.6: Rename a Remote Repository .....	149
<b>Chapter 46: Git Large File Storage (LFS)</b> .....	150
Section 46.1: Declare certain file types to store externally .....	150
Section 46.2: Set LFS config for all clones .....	150
Section 46.3: Install LFS .....	150
<b>Chapter 47: Git Patch</b> .....	151
Section 47.1: Creating a patch .....	151
Section 47.2: Applying patches .....	152
<b>Chapter 48: Git statistics</b> .....	153
Section 48.1: Lines of code per developer .....	153
Section 48.2: Listing each branch and its last revision's date .....	153
Section 48.3: Commits per developer .....	153
Section 48.4: Commits per date .....	154
Section 48.5: Total number of commits in a branch .....	154
Section 48.6: List all commits in pretty format .....	154
Section 48.7: Find All Local Git Repositories on Computer .....	154
Section 48.8: Show the total number of commits per author .....	154
<b>Chapter 49: git send-email</b> .....	155
Section 49.1: Use git send-email with Gmail .....	155
Section 49.2: Composing .....	155
Section 49.3: Sending patches by mail .....	155
<b>Chapter 50: Git GUI Clients</b> .....	157

<a href="#">Section 50.1: gitk and git-gui</a> .....	157
<a href="#">Section 50.2: GitHub Desktop</a> .....	158
<a href="#">Section 50.3: Git Kraken</a> .....	159
<a href="#">Section 50.4: SourceTree</a> .....	159
<a href="#">Section 50.5: Git Extensions</a> .....	159
<a href="#">Section 50.6: SmartGit</a> .....	159
<b>Chapter 51: Reflog - Restoring commits not shown in git log</b> .....	160
<a href="#">Section 51.1: Recovering from a bad rebase</a> .....	160
<b>Chapter 52: TortoiseGit</b> .....	161
<a href="#">Section 52.1: Squash commits</a> .....	161
<a href="#">Section 52.2: Assume unchanged</a> .....	162
<a href="#">Section 52.3: Ignoring Files and Folders</a> .....	164
<a href="#">Section 52.4: Branching</a> .....	165
<b>Chapter 53: External merge and difftools</b> .....	167
<a href="#">Section 53.1: Setting up KDiff3 as merge tool</a> .....	167
<a href="#">Section 53.2: Setting up KDiff3 as diff tool</a> .....	167
<a href="#">Section 53.3: Setting up an IntelliJ IDE as merge tool (Windows)</a> .....	167
<a href="#">Section 53.4: Setting up an IntelliJ IDE as diff tool (Windows)</a> .....	167
<a href="#">Section 53.5: Setting up Beyond Compare</a> .....	168
<b>Chapter 54: Update Object Name in Reference</b> .....	169
<a href="#">Section 54.1: Update Object Name in Reference</a> .....	169
<b>Chapter 55: Git Branch Name on Bash Ubuntu</b> .....	170
<a href="#">Section 55.1: Branch Name in terminal</a> .....	170
<b>Chapter 56: Git Client-Side Hooks</b> .....	171
<a href="#">Section 56.1: Git pre-push hook</a> .....	171
<a href="#">Section 56.2: Installing a Hook</a> .....	172
<b>Chapter 57: Git rerere</b> .....	173
<a href="#">Section 57.1: Enabling rerere</a> .....	173
<b>Chapter 58: Change git repository name</b> .....	174
<a href="#">Section 58.1: Change local setting</a> .....	174
<b>Chapter 59: Git Tagging</b> .....	175
<a href="#">Section 59.1: Listing all available tags</a> .....	175
<a href="#">Section 59.2: Create and push tag(s) in GIT</a> .....	175
<b>Chapter 60: Tidying up your local and remote repository</b> .....	177
<a href="#">Section 60.1: Delete local branches that have been deleted on the remote</a> .....	177
<b>Chapter 61: diff-tree</b> .....	178
<a href="#">Section 61.1: See the files changed in a specific commit</a> .....	178
<a href="#">Section 61.2: Usage</a> .....	178
<a href="#">Section 61.3: Common diff options</a> .....	178
<b>Credits</b> .....	179
<b>You may also like</b> .....	186