Contents

[Quick Guide To Revision Control 1](#_Toc289357273)

[Revision Control Methods for Documents 2](#_Toc289357274)

[Testing Repository At GitHub 5](#_Toc289357275)

[Global setup: 5](#_Toc289357276)

[Next steps: 5](#_Toc289357277)

[Existing Git Repo? 5](#_Toc289357278)

[Importing a Subversion Repo? 6](#_Toc289357279)

[Everyday GIT With 20 Commands Or So 6](#_Toc289357280)

[Individual Developer (Standalone) 6](#_Toc289357281)

[Examples 6](#_Toc289357282)

[Individual Developer (Participant) 7](#_Toc289357283)

[Examples 7](#_Toc289357284)

[Integrator 8](#_Toc289357285)

[Examples 9](#_Toc289357286)

[Repository Administration 10](#_Toc289357287)

[Examples 10](#_Toc289357288)

C

**General Comments about TCG Documents:**

* Terry reviewed several of the TNC documents, especially IF-TNCCS-SOH, in July 2010 as part of the white paper review of MS-SOH. Another quick review was performed last month. Except for IF-TNCCS-SOH, which is essentially MS-SOH with the introductory sections re-written, there did not appear to be a direct relationship between the TNC documents and the Microsoft NAP related documents. Microsoft does not use the standard TNC protocols (IF-T, IF-M, IF-IMC, IF-IMV, IF-MAP, IF-TNCCS, etc) in its implementation of NAP.
* The high-level architecture of TNC, NAC and NAP appear to be consistent. Each architecture has three computers. For NAP, there is the Windows Client, the PEP and the NAP Health Policy Server. For TNC, there is the Access Requestor, the PEP and the Policy Decision Point. Both NAP and TNC have three protocol layers: the Integrity Measurement Layer, the Integrity Evaluation Layer and the Network Access Layer. For NAP, those protocols are MS-WSH, MS-SOH and MS-DHCPN/MS-HCEP/MS-PEAP/MS-TSGU/MS-RNAP. For TNC, those protocols are IF-M, IF-TNCCS and IF-T.
* Most of the standards produced by TCG are purposely generic. One of the stated goals of TNC is to be vendor neutral. They are not going to document how any one vendor, such as Microsoft, implements any standard at a level of detail to allow a completely compatible implementation.
* TCG documents use a different set of terms than the Microsoft NAP documents. For example, while NAP uses the term NAP Client for the Windows client machine and NAP Health Policy Server for the Windows Server, TCG uses the terms Access Requestor (AR) and Policy Decision Point (PDP). While Microsoft uses the terms Security Health Agent (SHA) and Security Health Validator (SHV), TNC uses the terms Integrity Measurement Collectors and Integrity Measurement Verifiers. The difference in terminology makes it difficult to understand Microsoft NAP documentation in relation to TNC documents.
* The TCG documents introduce many concepts that have no correlation to the Microsoft NAP implementation. For example, the TNC Architecture describes Flow Controllers, Sensors and the Metadata Access Point Server, which do not seem to have an equivalent in NAP.

Gsdlgjslgjsdlgjlasdgjlsdgjlasdjglasdgjlsdkgklasdg

sdkdgjslagjaslkgjlasdgjlasdgjlasdgj

# Revision Control Methods for Documents

By A

**Git Release Notes (Git-1.7.4-preview20110204)**

Last update: 4 February 2011

**Introduction**

These release notes describe issues specific to the Git for Windows release.

General release notes covering the history of the core git commands are included in the subdirectory doc/git/html of the installation directory. Look for files starting with RelNotes.

See [Julie Melonihttp://git-scm.com/](http://chroniclegit-scm.com/blogs/profhacker/author/jmeloni/" \o "View all posts by Julie Meloni" /)

for further details about Git including ports to other operating systems. Git for Windows is hosted at [http://msysgit.googlecode.com/Julie Meloni](http://msysgit.googlecodechronicle.com/"/blogs/profhacker/author/jmeloni/" \o "View all posts by Julie Meloni).

**Known issues**

* Some commands are not yet supported on Windows and excluded from the installation; namely: git archimport, git cvsexportcommit, git cvsimport, git cvsserver, git instaweb, git shell.

The Logitec QuickCam software can cause spurious crashes. See "Why does make often crash creating a sh.exe.stackdump file when I try to compile my source code?" on the MinGW Wiki (

Both the concept and practice of revision control (also known as version control) are near and dear to my heart; a body of work as a technical writer, programmer, and project manager before moving over to academia made sure that particular personality trait was deeply ingrained. But during my time as a graduate student—when one might argue my sole purpose was to produce documents of one type or another.

**Revision Control is More Than Track Changes**

Although true that revision control systems track changes to written documents, there is much more to a revision control system than the visual cue of a strikethrough in a Microsoft Word document (or OpenOffice, etc). In those sorts of documents, when you accept changes (or reject them), once you save that document the revision history is gone (reset, if you will).

**An Example!** Suppose you send a document off to your committee chair/editor/writing group and you receive the document back full of suggested edits. Unless you manually saved a copy of the original (such as “article\_submitted.doc”) or the editor changed the name before sending it back, the new document will replace the old one of the same name. For a moment, assume that you did save a copy under a different name, so now you have “article\_submitted.doc” and “article\_edits.doc”. If you open “article\_edits.doc” and begin to work in it—accepting changes, making edits, and so on—what happens when you simply save it and close the file? In this case, “article\_edits.doc” has lost all of its original editorial comments; what you might have done is saved a file called “article\_edited.doc” or some such name, indicating you’ve begun editing the document based on the editorial suggestions.

“Well,” you might say, “I have the original file in my e-mail, so what does it matter? I can always get whatever version I need.” I would say, “Yes, you do. You also have at least three other files, which you have probably backed up in multiple places—and what, exactly, is managing those versions besides your own memory?” Nothing, is the answer.

While this process might work very well for you, and that’s great, it doesn’t really work for me—too many copies of files, too much duplication, too much to keep track of in my own pea brain. That’s when I remembered my technical writing background and began to apply it to my academic work.

**Simply Documenting Changes**

Perhaps you have seen business documents such as organizational charters or software manuals (or really anything in between) in which the first page of the document (or an appendix) includes a table like the following:

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision Number** | **Revision Date** | **Revision Notes** | **Owner** |
| 2.0 | 14 July 2010 | - final formatting changes - added Appendix C | Jane Doe |
| 1.9 | 10 July 2010 | - added coverage of Widget X - removed paragraph about Widget Z | Jane Doe |

* Typically, these revision notes will go hand in hand with a revision control system employed by the writer or organization. With such a system in place, there would only ever be one document in the repository, regardless of how many revisions had been performed. And, more importantly, all of the versions would be accessible at least for viewing, if not checking out or reverting completely. By that I mean if the mythical Jane Doe’s company decided to resume production of Widget Z, Jane could simply view version 1.8 of the document through her version control software, find the chunk that had been removed (either by looking, or by performing a file comparison ([diffhttp://www.mingw.org/wiki/Environment\_issuesdiff](http://enwww.mingwen.wikipedia.org/wiki/Environment_issues"Diff))),)
* The Quick Launch icon will only be installed for the user running setup (typically the Administrator). This is a technical restriction and will not change.
* curl uses $HOME/\_netrc instead of $HOME/.netrc.
* If you want to specify a different location for --upload-pack, you have to start the absolute path with two slashes. Otherwise MSys will mangle the path.
* git and bash have serious problems with non-ASCII file names (Issue 80, 159).
* If configured to use plink, you will have to connect with putty first and accept the host key.
* As merge tools are executed using the MSys bash, options starting with "/" need to be handled specially: MSys would interpret that as a POSIX path, so you need to double the slash (Issue 226). Example: instead of "/base", say "//base". Also, extra care has to be paid to pass Windows programs Windows paths, as they have no clue about MSys style POSIX paths -- You can use something like $(cmd //c echo "$POSIXPATH").

**Changes since Git-1.7.3.2-preview20101025**

***New Features***

* Comes with Git 1.7.4 plus patches.
* Includes antiword to enable viewing diffs of .doc files
* Includes poppler to enable viewing diffs of .pdf files
* Removes cygwin paths from the bash shell PATH

***Bugfixes***

* Please refer to the release notes for official Git 1.7.4

**Changes since Git-1.7.3.1-preview20101002**

***New Features***

* Comes with Git 1.7.3.2 plus patches.

**Changes since Git-1.7.2.3-preview20100911**

***New Features***

* Comes with Git 1.7.3.1 plus patches.
* Updated to Vim 7.3, file-5.04 and InnoSetup 5.3.11

***Bugfixes***

* Issue 528 (remove uninstaller from Start Menu) was fixed
* Issue 527 (failing to find the certificate authority bundle) was fixed
* Issue 524 (remove broken and unused sdl-config file) was fixed
* Issue 523 (crash pushing to WebDAV remote) was fixed

**Changes since Git-1.7.1-preview20100612**

***New Features***

* Comes with Git 1.7.2.3 plus patches.

***Bugfixes***

* Issue 519 (build problem with compat/regex/regexec.c) was fixed
* Issue 430 (size of panes not preserved in git-gui) was fixed
* Issue 411 (git init failing to work with CIFS paths) was fixed
* Issue 501 (failing to clone repo from root dir using relative path) was fixed

**Changes since Git-1.7.0.2-preview20100309**

***New Features***

* Comes with Git 1.7.1 plus patches.

***Bugfixes***

* Issue 27 (git-send-mail not working properly) was fixed again
* Issue 433 (error while running git svn fetch) was fixed
* Issue 427 (Gitk reports error: "couldn't compile regular expression pattern: invalid repetition count(s)") was fixed
* Issue 192 (output truncated) was fixed again
* Issue 365 (Out of memory? mmap failed) was fixed
* Issue 387 (gitk reports "error: couldn't execute "git:" file name too long") was fixed
* Issue 409 (checkout of large files to network drive fails on XP) was fixed
* Issue 428 (The return value of git.cmd is not the same as git.exe) was fixed
* Issue 444 (Git Bash Here returns a "File not found error" in Windows 7 Professional - 64 bits) was fixed
* Issue 445 (git help does nothing) was fixed
* Issue 450 ("git --bare init" shouldn't set the directory to hidden.) was fixed
* Issue 456 (git script fails with error code 1) was fixed
* Issue 469 (error launch wordpad in last netinstall) was fixed
* Issue 474 (git update-index --index-info silently does nothing) was fixed
* Issue 482 (Add documentation to avoid "fatal: $HOME not set" error) was fixed
* Issue 489 (git.cmd issues warning if %COMSPEC% has spaces in it) was fixed
* Issue 436 ("mkdir : No such file or directory" error while using git-svn to fetch or rebase) was fixed
* Issue 440 (Uninstall does not remove cheetah.) was fixed
* Issue 441 (Git-1.7.0.2-preview20100309.exe installer fails with unwritable msys-1.0.dll when ssh-agent is running) was fixed

**Changes since Git-1.6.5.1-preview20091022**

***New Features***

* Comes with official Git 1.7.0.2.
* Comes with Git-Cheetah (on 32-bit Windows only, for now).
* Comes with connect.exe, a SOCKS proxy.
* Tons of improvements in the installer, thanks to Sebastian Schuberth.
* On Vista, if possible, symlinks are used for the built-ins.
* Features Hany's dos2unix tool, thanks to Sebastian Schuberth.

Updated Tcl/Tk to version 8.5.8 (thanks Pat Thoyts!).)), and add it back into the current of the document without wrecking the revision history or creating duplicate files “littering” her hard drive.

**But I’m an academic! I don’t need to document widgets!** While perhaps true, I’ll bet you’ve revised and/or repurposed a document before. Some of you might do such things on a regular basis, especially if you’re involved in long-term projects involving groups of people and funding—there are activities, results, and plans to document on a regular basis. Perhaps you can keep all these document revisions in your head, and even know what text you’ve added or cut, and when, and know from where you can reclaim that text if something goes horribly wrong (or someone just says “let’s add that back in…”). I can’t.

With a single document and a version control process in place, you could work continuously/incrementally on, say, “Year 1 Plan” until it has been finalized; the version control software and notes made with each version would provide a snapshot of changes and a quick way to view, verify, and recover content that has been added or modified. If a team member says something like “didn’t we define that task in the plan? I don’t see it” and you have to counter with “yes, but then we took it out after a meeting in April,” the conversation doesn’t have to stop there—and you don’t have to try to recreate that task description from memories of snippets of meetings gone by. Instead, you can simply navigate through your repository, find the version that has with it a note to the effect of “removed task X description”, view it, grab the text, and either insert it back into the document or send it forward for more discussion. When it’s time to begin work on the “Year 2 Plan” you can just start with the final version of “Year 1 Plan” and start a new branch of the “project”.

The point is this: nothing is lost—not a document, not a suggested edit to a document, and not a piece of text once added then removed. You can recover anything in a few clicks because you employed revision control methods from the start. So, when your dissertation advisor says “do you remember that section of Chapter 3 that we talked about three months ago, that we decided to remove entirely? Let’s put that back in and expand the examples,” you don’t have a heart attack and begin the process of recreating from memory something that you stopped thinking about months ago (or, you don’t begin the process of sifting through manually created versions of documents called “diss\_ch3\_1.doc,” “diss\_ch3\_5.doc,” “diss\_ch3\_9.doc,” and so on).

**Create a Personal Repository**

* You can begin the process of creating a personal document repository simply by installing a Subversion client on your own machine—no external server necessary (this also means you’re the only one who can checkin/checkout documents, but this post has been about you and your own work anyway, so we’ll save the rest for another day). For Windows users,
* default, only .git/ is hidden, to work around a bug in Eclipse (thanks to Erik Faye-Lund).

***Bugfixes***

* Fixed threaded grep (thanks to Heiko Voigt).
* git gui was fixed for all kinds of worktree-related failures (thanks Pat Thoyts).
* git gui now fully supports themed widgets (thanks Pat Thoyts and Heiko Voigt).
* Git no longer complains about an unset RUNTIME\_PREFIX (thanks Johannes Sixt).
* git gui can Explore Working Copy on Windows again (thanks Markus Heidelberg).
* git gui can create shortcuts again (fixes issue 425, thanks Heiko Voigt).
* When "git checkout" cannot overwrite files because they are in use, it will offer to try again, giving the user a chance to release the file (thanks Heiko Voigt).
* Ctrl+W will close gitk (thanks Jens Lehmann).
* git gui no longer binds Ctrl+C, which caused problems when trying to use said shortcut for the clipboard operation "Copy" (fixes issue 423, thanks Pat Thoyts).
* gitk does not give up when the command line length limit is reached (issue 387).
* The exit code is fixed when Git.cmd is called from cmd.exe (thanks Alexey Borzenkov).
* When launched via the (non-Cheetah) shell extension, the window icon is now correct (thanks Sebastian Schuberth).
* Uses a TrueType font for the console, to be able to render UTF-8 correctly.
* Clarified the installer's line ending options (issue 370).
* Substantially speeded up startup time from cmd unless NO\_FSTAB\_THREAD is set (thanks Johannes Sixt).
* Update msys-1.0.dll yet again, to handle quoted parameters better (thanks Heiko Voigt).
* Updated cURL to a version that supports SSPI.
* Updated tar to handle the pax headers generated by *git archive*.
* Updated sed to a version that can handle the filter-branch examples.
* .git\* files can be associated with the default text editor (issue 397).

**Changes since Git-1.6.4-preview20090729**

***New Features***

* Comes with official git 1.6.5.1.
* Thanks to Johan 't Hart, files and directories starting with a single dot (such as '.git') will now be marked hidden (you can disable this setting with core.hideDotFiles=false in your config) (Issue 288).
* Thanks to Thorvald Natvig, Git on Windows can simulate symbolic links by using reparse points when available. For technical reasons, this only works for symbolic links pointing to files, not directories.
* A lot of work has been put into making it possible to compile Git's source code (the part written in C, of course, not the scripts) with Microsoft Visual Studio. This work is ongoing.
* Thanks to Sebastian Schuberth, we only offer the (Tortoise)Plink option in the installer if the presence of Plink was detected and at least one Putty session was found..
* Thanks to Sebastian Schuberth, the installer has a nicer icon now.
* Some more work by Sebastian Schuberth was done on better integration of Plink (Issues 305 & 319).

***Bugfixes***

* Thanks to Sebastian Schuberth, *git svn* picks up the SSH setting specified with the installer (Issue 305).

**Changes since Git-1.6.3.2-preview20090608**

***New Features***

* Comes with official git 1.6.4.
* Supports [https://TortoiseSVN](https://"http://tortoisesvn.tigris.org/) URLs, thanks to Erik Faye-Lund.
* Supports send-email, thanks to Erik Faye-Lund (Issue 27).
* Updated Tcl/Tk to version 8.5.7, thanks to Pat Thoyts.

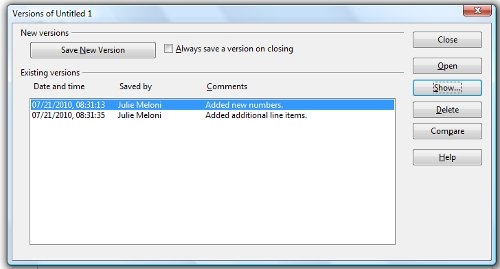
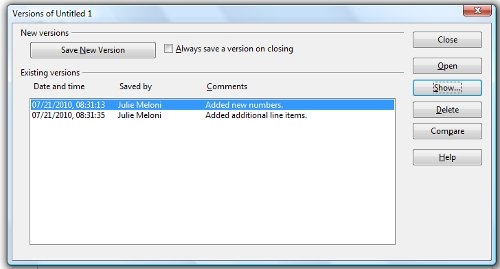
***Bugfixes***

The home (free, open source) is very easy to install and use; you can create a repository in just a few minutes. For Mac users (actually, Windows and Linux too), [RapidSVN](http://www.rapidsvn.org/index.php/Documentation) (free, open source) is a good client; like TortoiseSVN it offers a pleasant (read: not a command-line interface which tends to scare people away) graphical user interface to the underlying version control system.

I will save the “Installing and Using SVN on Your Own Machine” post for another day, as the goal here was to get you thinking about the processes and how revision control might fit into your own writing practices, **but** if you would like to get started and fiddle around creating repositories on your own machine I recommend reading [“Subversion for Writers”](http://strangenoises.org/subversion-for-writers/) (Mac examples) and [” Getting Started With Subversion—Part 1: The Basics”](http://www.oak-tree.us/blog/index.php/2009/02/13/subversion1) (Windows examples).

**Version Control in OpenOffice**

If you use [OpenOffice](http://chronicle.com/blogPost/Considering-OpenOffice-You/22850/), you have quick access to a rudimentary version control system just by accessing “Versions” under the File menu. on the dialog box shown below, you can see the notes I have left for myself in the revision history of a particular document. I can perform a diff (compare) between two versions, open a previous version as a read-only document, or open a previous version for writing (and thereby possibly create a fork in my document creation).



Opening and working with revisions in this way in OpenOffice might help the concepts crystallize before moving on to installing additional software or working with a decentralized model used by a team. If you already use OpenOffice, it sure can’t hurt.

**Use Revision Control For Good**

In this case, I’m using “good” to mean “productive”. I recently had occasion to review some of my own writing, and I knew in the back of my mind that somewhere along the line I had cut large swaths of text yet couldn’t for the life of me remember (or find) the file called “stuff\_i\_cut\_might\_be\_important\_but\_not\_now.doc”—if ever I had made one to begin with. Had I used version control, I could simply have gone through my notes and found it in a previous version of the document. The time I wasted first looking for the content and then trying to recreate the content certainly wasn’t productive.

Before I write the next post related to version control, **in which other cases can you imagine document revision control being useful to you? Related to this topic, which programs, services, and processes interest you most?** Let us know in the comments. Or, just work through your own scenario and get feedback as to whether or not implementing revision control could help you navigate a roadblock or other hurdle.

# sitory Everyday GIT With 20 Commands Or So

[[Individual Developer (Standalone)]](http://www.kernel.org/pub/software/scm/git/docs/everyday.html" \l "Individual%20Developer%20%28Standalone%29) commands are essential for anybody who makes a commit, even for somebody who works alone.

If you work with other people, you will need commands listed in the [[Individual Developer (Participant)]](http://www.kernel.org/pub/software/scm/git/docs/everyday.html" \l "Individual%20Developer%20%28Participant%29) section as well.

People who play the [[Integrator]](http://www.kernel.org/pub/software/scm/git/docs/everyday.html" \l "Integrator) role need to learn some more commands in addition to the above.

[[Repository Administration]](http://www.kernel.org/pub/software/scm/git/docs/everyday.html" \l "Repository%20Administration) commands are for system administrators who are responsible for the care and feeding of git repositories.

## Individual Developer (Standalone)

A standalone individual developer does not exchange patches with other people, and works alone in a single repository, using the following commands.

* [git-init(1)](http://www.kernel.org/pub/software/scm/git/docs/git-init.html) to create a new repository.
* [git-show-branch(1)](http://www.kernel.org/pub/software/scm/git/docs/git-show-branch.html) to see where you are.
* [git-log(1)](http://www.kernel.org/pub/software/scm/git/docs/git-log.html) to see what happened.
* [git-checkout(1)](http://www.kernel.org/pub/software/scm/git/docs/git-checkout.html) and [git-branch(1)](http://www.kernel.org/pub/software/scm/git/docs/git-branch.html) to switch branches.
* [git-add(1)](http://www.kernel.org/pub/software/scm/git/docs/git-add.html) to manage the index file.
* [git-diff(1)](http://www.kernel.org/pub/software/scm/git/docs/git-diff.html) and [git-status(1)](http://www.kernel.org/pub/software/scm/git/docs/git-status.html) to see what you are in the middle of doing.
* [git-commit(1)](http://www.kernel.org/pub/software/scm/git/docs/git-commit.html) to advance the current branch.
* [git-reset(1)](http://www.kernel.org/pub/software/scm/git/docs/git-reset.html) and [git-checkout(1)](http://www.kernel.org/pub/software/scm/git/docs/git-checkout.html) (with pathname parameters) to undo changes.
* [git-merge(1)](http://www.kernel.org/pub/software/scm/git/docs/git-merge.html) to merge between local branches.
* [git-rebase(1)](http://www.kernel.org/pub/software/scm/git/docs/git-rebase.html) to maintain topic branches.
* [git-tag(1)](http://www.kernel.org/pub/software/scm/git/docs/git-tag.html) to mark known point.

### Examples

Use a tarball as a starting point for a new repository.

$ tar zxf frotz.tar.gz

$ cd frotz

$ git init

$ git add . **<1>**

$ git commit -m "import of frotz source tree."

$ git tag v2.43 **<2>**

* add everything under the current directory. is now discovered properly (Issues 108 & 259)..
* IPv6 is supported now, thanks to Martin Martin Storsjö (Issue 182).

**Changes since Git-1.6.3-preview20090507**

***New Features***

* Comes with official git 1.6.3.2.
* Uses TortoisePlink instead of Plink if available.

***Bugfixes***

* Plink errors out rather than hanging when the user needs to accept a host key first (Issue 96).
* The user home directory is inferred from $HOMEDRIVE\$HOMEPATH instead of $HOME (Issue 108).
* The environment setting $CYGWIN=tty is ignored (Issues 138, 248 and 251).
* The "ls" command shows non-ASCII filenames correctly now (Issue 188).
* Adds more syntax files for vi (Issue 250).
* $HOME/.bashrc is included last from /etc/profile, allowing .bashrc to override all settings in /etc/profile (Issue 255).
* Completion is case-insensitive again (Issue 256).
* The "start" command can handle arguments with spaces now (Issue 258).
* For some Git commands (such as "git commit"), vi no longer "restores" the cursor position.

**Changes since Git-1.6.2.2-preview20090408**

***New Features***

* Comes with official git 1.6.3.
* Thanks to Marius Storm-Olsen, Git has a substantially faster readdir() implementation now.
* Marius Storm-Olsen also contributed a patch to include nedmalloc, again speeding up Git noticably.
* Compiled with GCC 4.4.0

***Bugfixes***

* Portable Git contains a README.portable.
* Portable Git now actually includes the builtins.
* Portable Git includes git-cmd.bat and git-bash.bat.
* Portable Git is now shipped as a .7z; it still is a self-extracting archive if you rename it to .exe.
* Git includes the Perl Encode module now.

1. Git now includes the filter-make a lightweight, unannotated tag.

* Create a topic branch tool.and develop.
* There is a workaround for a Windows 7 regression triggering a crash in the progress reporting (e.g. during a clone). This fixes issues 236 and 247.
* gitk tries not to crash when it is closed while reading references (Issue 125, thanks Pat Thoyts).
* In some setups, hard-linking is not as reliable as it should be, so we have a workaround which avoids hard links in some situations (Issues 222 and 229).
* git-svn sets core.autocrlf to false now, hopefully shutting up most of the git-svn reports.

**Changes since Git-1.6.2.1-preview20090322**

***New Features***

* Comes with official git 1.6.2.2.
* Upgraded Tcl/Tk to 8.5.5.
* TortoiseMerge is supported by mergetool now.
* Uses pthreads (faster garbage collection on multi-core machines).

The $ git checkout -b alsa-audio **<1>**

* $ edit/compile/test suite passes!

***Bugfixes***

* Renaming was made more robust (due to Explorer or some virus scanners, files could not be renamed at the first try, so we have to try multiple times).
* Johannes Sixt made lots of changes to the test-suite to identify properly which tests should pass, and which ones cannot pass due to limitations of the platform.
* Support PAGERs with spaces in their filename.

Quite a few$ git checkout -- curses/ux\_audio\_oss.c **<2>**

$ git add curses/ux\_audio\_alsa.c **<3>**

$ edit/compile/test

$ git diff HEAD **<4>**

$ git commit -a -s **<5>**

$ edit/compile/test

$ git reset --soft HEAD^ **<6>**

$ edit/compile/test

$ git diff ORIG\_HEAD **<7>**

$ git commit -a -c ORIG\_HEAD **<8>**

$ git checkout master **<9>**

$ git merge alsa-audio **<10>**

$ git log --since='3 days ago' **<11>**

$ git log v2.43.. curses/ **<12>**

1. create a new topic branch.
2. revert your botched changes in curses/ux\_audio\_oss.c.
3. you need to tell git if you added a new file; removal and modification will be caught if you do git commit -a later.
4. to see what changes youwereyou are committing.
5. commit everything as you have tested, with your sign-off.
6. take the last commit back, keeping what is in the working tree.
7. look at the changes since the premature commit we took back.
8. redo the commit undone which we needed in the previous step, using the message you originally wrote.
9. switch to the master branch.
10. merge a topic branch into your master branch.
11. review commit logs; other forms to limit output can be combined and include --max-count=10 (show 10 commits), --until=2005-12-10, etc.
12. view only the changes that touch what’s in curses/ directory, since v2.43 tag.

## Individual Developer (Participant)

* A developer working as a participant in athe olden days of msysGita group project needs to learn how to communicate with others, and uses these commands in addition to the ones needed by a standalone developer.
* Fall back to / when HOME cannot be set to the real home directory due to locale issues (works around Issue 108 for the moment).

**Changes since Git-1.6.2-preview20090308**

***New Features***

* Comes with official git 1.6.2.1.
* A portable application is shipped in addition to the installer (Issue 195).
* Comes with a Windows-specific mmap() implementation (Issue 198).

***Bugfixes***

* ANSI control characters are no longer shown verbatim (Issue 124).
* Temporary files are created respecting core.autocrlf (Issue 177).
* The Git Bash prompt is colorful again (Issue 199).
* Fixed crash when hardlinking during a clone failed (Issue 204).
* An infinite loop was fixed in git-gui (Issue 205).
* The ssh[git-clone(1)](http://www.kernel.org/pub/software/scm/git/docs/git-clone.html) from the upstream to prime your local repository.
* [git-pull(1)](http://www.kernel.org/pub/software/scm/git/docs/git-pull.html) and [git-fetch(1)](http://www.kernel.org/pub/software/scm/git/docs/git-fetch.html) from "origin" to keep up-to-date with the upstream.
* [git-push(1)](http://www.kernel.org/pub/software/scm/git/docs/git-push.html) to shared repository, if you adopt CVS style shared repository workflow.
* [git-format-patch(1)](http://www.kernel.org/pub/software/scm/git/docs/git-format-patch.html) to prepare e-mail submission, if you adopt Linux kernel-style public forum workflow.

### Examples

Clone the upstream and work on it. Feed changes to upstream.

$ git clone git://git.kernel.org/pub/scm/.../torvalds/linux-2.6 my2.6

$ cd my2.6

$ edit/compile/test; git commit -a -s **<1>**

$ git format-patch origin **<2>**

$ git pull **<3>**

$ git log -p ORIG\_HEAD.. arch/i386 include/asm-i386 **<4>**

$ git pull git://git.kernel.org/pub/.../jgarzik/libata-dev.git ALL **<5>**

$ git reset --hard ORIG\_HEAD **<6>**

$ git gc **<7>**

$ git fetch --tags **<8>**

1. repeat as needed.
2. extract patches from your branch for e-mail submission.
3. git pull fetches from origin by default and merges into the current branch.
4. immediately after pulling, look at the changes done upstream since last time we checked, only in the area we are interested in.
5. fetch from a specific branch from a specific repository and merge.
6. revert the pull.
7. garbage collect leftover objects from reverted pull.
8. from time to time, obtain official tags from the origin and store them under .git/refs/tags/.

Push into another repository.

satellite$ git clone mothership:frotz frotz **<1>**

satellite$ cd frotz

satellite$ git config --get-regexp '^(remote|branch)\.' **<2>**

remote.origin.url mothership:frotz

remote.origin.fetch refs/heads/\*:refs/remotes/origin/\*

branch.master.remote origin

branch.master.merge refs/heads/master

satellite$ git config remote.origin.push \

master:refs/remotes/satellite/master **<3>**

satellite$ edit/compile/test/commit

satellite$ git push origin **<4>**

mothership$ cd frotz

mothership$ git checkout master

mothership$ git merge satellite/master **<5>**

1. mothership machine has a frotz repository under your home directory; clone from it to start a repository on the satellite machine.
2. clone sets these configuration variables by default. It arranges git pull to fetch and store the branches of mothership machine to local remotes/origin/\* remote-tracking branches.
3. arrange git push to push local master branch to remotes/satellite/master branch of the mothership machine.
4. push will stash our work away on remotes/satellite/master remote-tracking branch on the mothership machine. You could use this as a back-up method.
5. on mothership machine, merge the work done on the satellite machine into the master branch.

Branch off of a specific tag.

$ git checkout -b private2.6.14 v2.6.14 **<1>**

$ edit/compile/test; git commit -a

$ git checkout master

$ git format-patch -k -m --stdout v2.6.14..private2.6.14 |

git am -3 -k **<2>**

1. create a private branch based on a well known (but somewhat behind) tag.
2. forward port all changes in private2.6.14 branch to master branch without a formal "merging".

## Integrator

A fairly central person acting as the integrator in a group project receives changes made by others, reviews and integrates them and publishes the result for others to use, using these commands in addition to the ones needed by participants.

* [git-am(1)](http://www.kernel.org/pub/software/scm/git/docs/git-am.html) to apply patches e-mailed in from your contributors.
* [git-pull(1)](http://www.kernel.org/pub/software/scm/git/docs/git-pull.html) to merge from your trusted lieutenants.
* [git-format-patch(1)](http://www.kernel.org/pub/software/scm/git/docs/git-format-patch.html) to prepare and send suggested alternative to contributors.
* [git-revert(1)](http://www.kernel.org/pub/software/scm/git/docs/git-revert.html) to undo botched commits.
* [git-push(1)](http://www.kernel.org/pub/software/scm/git/docs/git-push.html) to publish the bleeding edge.

### Examples

My typical GIT day.

$ git status **<1>**

$ git show-branch **<2>**

$ mailx **<3>**

& s 2 3 4 5 ./+to-apply

& s 7 8 ./+hold-linus

& q

$ git checkout -b topic/one master

$ git am -3 -i -s -u ./+to-apply **<4>**

$ compile/test

$ git checkout -b hold/linus && git am -3 -i -s -u ./+hold-linus **<5>**

$ git checkout topic/one && git rebase master **<6>**

$ git checkout pu && git reset --hard next **<7>**

$ git merge topic/one topic/two && git merge hold/linus **<8>**

$ git checkout maint

$ git cherry-pick master~4 **<9>**

$ compile/test

$ git tag -s -m "GIT 0.99.9x" v0.99.9x **<10>**

$ git fetch ko && git show-branch master maint 'tags/ko-\*' **<11>**

$ git push ko **<12>**

$ git push ko v0.99.9x **<13>**

1. see what I was in the middle of doing, if any.
2. see what topic branches I have and think about how ready they are.
3. read mails, save ones that are applicable, and save others that are not quite ready.
4. apply them, interactively, with my sign-offs.
5. create topic branch as needed and apply, again with my sign-offs.
6. rebase internal topic branch that has not been merged to the master, nor exposed as a part of a stable branch.
7. restart pu every time from the next.
8. and bundle topic branches still cooking.
9. backport a critical fix.
10. create a signed tag.
11. make sure I did not accidentally rewind master beyond what I already pushed out. ko shorthand points at the repository I have at kernel.org, and looks like this:
12. $ cat .git/remotes/ko
13. URL: kernel.org:/pub/scm/git/git.git
14. Pull: master:refs/tags/ko-master
15. Pull: next:refs/tags/ko-next
16. Pull: maint:refs/tags/ko-maint
17. Push: master
18. Push: next
19. Push: +pu

Push: maint

In the output from git show-branch, master should have everything ko-master has, and next should have everything ko-next has.

1. push out the bleeding edge.
2. push the tag out, too.

## Repository Administration

A repository administrator uses the following tools to set up and maintain access to the repository by developers.

* [git-daemon(1)](http://www.kernel.org/pub/software/scm/git/docs/git-daemon.html) to allow anonymous download from repository.
* [git-shell(1)](http://www.kernel.org/pub/software/scm/git/docs/git-shell.html) can be used as a restricted login shell for shared central repository users.

[update hook howto](http://www.kernel.org/pub/software/scm/git/docs/howto/update-hook-example.txt) has a good example of managing a shared central repository.

### Examples

We assume the following in /etc/services

$ grep 9418 /etc/services

git 9418/tcp # Git Version Control System

Run git-daemon to serve /pub/scm from inetd.

$ grep git /etc/inetd.conf

git stream tcp nowait nobody \

/usr/bin/git-daemon git-daemon --inetd --export-all /pub/scm

The actual configuration line should be on one line.

Run git-daemon to serve /pub/scm from xinetd.

$ cat /etc/xinetd.d/git-daemon

# default: off

# description: The git server offers access to git repositories

service git

{

disable = no

type = UNLISTED

port = 9418

socket\_type = stream

wait = no

user = nobody

server = /usr/bin/git-daemon

server\_args = --inetd --export-all --base-path=/pub/scm

log\_on\_failure += USERID

}

Check your xinetd(8) documentation and setup, this is from a Fedora system. Others might be different.

Give push/pull only access to developers.

$ grep git /etc/passwd **<1>**

alice:x:1000:1000::/home/alice:/usr/bin/git-shell

bob:x:1001:1001::/home/bob:/usr/bin/git-shell

cindy:x:1002:1002::/home/cindy:/usr/bin/git-shell

david:x:1003:1003::/home/david:/usr/bin/git-shell

$ grep git /etc/shells **<2>**

/usr/bin/git-shell

1. log-in shell is set to /usr/bin/git-shell, which does not allow anything but git push and git pull. The users should get an ssh access to the machine.
2. in many distributions /etc/shells needs to list what is used as the login shell.

CVS-style shared repository.

$ grep git /etc/group **<1>**

git:x:9418:alice,bob,cindy,david

$ cd /home/devo.git

$ ls -l **<2>**

lrwxrwxrwx 1 david git 17 Dec 4 22:40 HEAD -> refs/heads/master

drwxrwsr-x 2 david git 4096 Dec 4 22:40 branches

-rw-rw-r-- 1 david git 84 Dec 4 22:40 config

-rw-rw-r-- 1 david git 58 Dec 4 22:40 description

drwxrwsr-x 2 david git 4096 Dec 4 22:40 hooks

-rw-rw-r-- 1 david git 37504 Dec 4 22:40 index

drwxrwsr-x 2 david git 4096 Dec 4 22:40 info

drwxrwsr-x 4 david git 4096 Dec 4 22:40 objects

drwxrwsr-x 4 david git 4096 Nov 7 14:58 refs

drwxrwsr-x 2 david git 4096 Dec 4 22:40 remotes

$ ls -l hooks/update **<3>**

-r-xr-xr-x 1 david git 3536 Dec 4 22:40 update

$ cat info/allowed-users **<4>**

refs/heads/master alice\|cindy

refs/heads/doc-update bob

refs/tags/v[0-9]\* david

1. place the developers into the same git group.
2. and make the shared repository writable by the group.
3. use update-hook example by Carl from Documentation/howto/ for branch policy control.
4. alice and cindy can push into master, only bob can push into doc-update. david is the release manager and is the only person who can create and push version tags.

* HTTP server to support dumb protocol is always used with plink.exe (Issue 209).transfer.
* devMore vim files are shipped now, so that syntax highlighting works.

**Changes since Git-1.6.1-preview20081225**

***New Features***

* Comes with officialdev$ git 1.6.2.update-server-info **<1>**
* devComes with upgraded vim 7.2.
* Compiled with GCC 4.3.3.
* Thedev$ ftp user can choose the preferred CR/LF behavior in the installer now.
* Peter Kodl contributed support for hardlinks on Windows.
* The bash prompt shows information about the current repository.

***Bugfixes***

* If supported by the file system, pack files can grow larger than 2gb.
* Comes with updated msys-1.0.dll (should fix some Vista issues).
* Assorted fixes to support the new libexec/git-core/ layout better.
* Read-only files can be properly replaced now.
* git-svn is included again (original caveats still apply).
* Obsolete programs from previous installations are cleaned up.

**Changes since Git-1.6.0.2-preview20080923**

***New Features***

* Comes with official git 1.6.1.
* Avoid useless console windows.
* Installer remembers how to handle PATH.

**Changes since Git-1.6.0.2-preview20080921**

***Bugfixes***

* ssh works again.
* 'git add -p' works again.
* Various programs that aborted with 'Assertion failed: argv0\_path' are fixed.

**Changes since Git-1.5.6.1-preview20080701**

***Removed Features***

* git svn is excluded from the end-user installer (see Known Issues).

***New Features***

* Comes with official git 1.6.0.2.

***Bugfixes***

* No Windows-specific bugfixes.

**Changes since Git-1.5.6-preview20080622**

***New Features***

* Comes with official git 1.5.6.1.

***Bugfixes***

* Includes fixed msys-1.0.dll that supports Vista and Windows Server 2008 (Issue 122).
* cmd wrappers do no longer switch off echo.

**Changes since Git-1.5.5-preview20080413**

***New Features***

* Comes with official git 1.5.6.
* Installer supports configuring a user provided plink (PuTTY).

***Bugfixes***

* Comes with tweaked msys-1.0.dll to solve some command line mangling issues.
* cmd wrapper does no longer close the command window.
* Programs in the system PATH, for @isp.example. editors, can be launched from Git without specifying their full path..com **<2>**
* "ftp> cp -r .git stash apply stash@{1}" works.
* Comes with basic ANSI control code emulation for the Windows console to avoid wrapping of pull/merge's diffstats.
* Git correctly passes port numbers to PuTTY's plink

**Changes since Git-1.5.4-preview20080202**

***New Features***

* Comes with official git 1.5.5.
* core.autocrlf is enabled (true) by default. This means git converts to Windows line endings (CRLF) during checkout and converts to Unix line endings (LF) during commit. This is the right choice for cross-platform projects. If the conversion is not reversible, git warns the /home/user/. The installer warns about the new default before the installation starts./myproject.git
* makeThe user does no longer have to "accept" the GPL but only needs to press "continue".
* Installer deletes shell scripts that have been replaced by builtins. Upgrading should be safer.
* Supports "git svn". Note that the performance might be belowmake sure your infoexpectation.

***Bugfixes***

* Newer ssh fixes connection failures (issue 74).
* Comes with MSys-1.0.11-20071204. This should solve some "fork: resource unavailable" issues.
* All DLLsinfo/refs and objects/info/packs are up-rebased to avoid problems with "fork" on Vista.

**Changes since Git-1.5.3.6-preview20071126**

***New Features***

* Comes with official git 1.5.4.
* Some commands that are not yet suppoted on Windows are no longer included (see Known Issues above).
* Release notes are displayed in separate window.
* Includes qsort replacement to improve performance on Windows 2000.

***Bugfixes***

* Fixes invalid error message that setup.ini cannot be deleted on uninstall.
* Setup tries harder to finish the installation and reports more detailed errors.
* Vim's syntax highlighting is suitable for dark background.

**Changes since Git-1.5.3.5-preview20071114**

***New Features***

* Git is included in version 1.5.3.6.
* Setup displays release notes.

***Bugfixes***

* pull/fetch/push in git-gui works. Note, there is no way for ssh to ask for a passphrase or for confirmation if you connect to an unknown host. So, you must have ssh set up to work without passphrase. Either you have a key without passphrase, or you started ssh-agent. You may also consider using PuTTY by pointing GIT\_SSH to plink.exe and handle your ssh keys with Pageant. In this case you should include your login name in urls. You must also connect to an unknown host once from the command line and confirm the host key, before you can use it from git-gui.-to-date

**Changes since Git-1.5.3-preview20071027**

***New Features***

* Git is included in version 1.5.3.5.
* Setup can be installed as normal user.
* When installing as Administrator, all icons except the Quick Launch icon will be created for all users.
* "git help user-manual" displays the user manual.

***Bugfixes***

* Git Bash works on Windows XP 64.

**Changes since Git-1.5.3-preview20071019**

***Bugfixes***

* The templates for a new repository are found.
* The global configuration /etc/gitconfig is found.
* Git Gui localization works. It falls back to English if a translation has errors.

**Changes since WinGit-0.2-alpha**

1. The history of the release notes stops here. Various new features and bugfixes are available since WinGit-0.2-alpha. Please check the git history of the msysgit project for details.upload to public HTTP server hosted by your ISP.