

SECOND EDITION

---

# bash Cookbook

## *Solutions and Examples for bash Users*

This excerpt contains Appendix D, extracted for ease of reference and to encourage using revision control for your bash coding. The complete book is available on the O'Reilly Online Learning Platform and through other retailers.

*Carl Albing and JP Vossen*

Beijing • Boston • Farnham • Sebastopol • Tokyo

**O'REILLY®**

## **bash Cookbook**

by Carl Albing and JP Vossen

Copyright © 2018 Carl Albing and JP Vossen. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreilly.com/safari>). For more information, contact our corporate/institutional sales department: 800-998-9938 or [corporate@oreilly.com](mailto:corporate@oreilly.com).

**Editor:** Jeff Bleiel

**Production Editor:** Kristen Brown

**Copyeditor:** Rachel Head

**Proofreader:** James Fraleigh

**Indexer:** Ellen Troutman-Zaig

**Interior Designer:** David Futato

**Cover Designer:** Karen Montgomery

**Illustrator:** Rebecca Demarest

May 2007:           First Edition  
October 2017:      Second Edition

### **Revision History for the Second Edition**

2017-10-04:   First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781491975336> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *bash Cookbook*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

While the publisher and the authors have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the authors disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-491-97533-6

[LSI]

---

# Revision Control

Revision control systems are a way not only to travel back in time, but to see what has changed at various points in your timeline. They are also called *versioning* or *version control* systems, which is actually a more technically accurate name. Such a system allows you to maintain a *repository* of files in a project, and to keep track of changes to those files, as well as the reasons for those changes. Modern revision control systems allow more than one developer to work concurrently on the same project, or even the same file.

Revision control systems are essential to modern software development efforts, but they are also useful in many other areas, such as writing documentation, tracking system configurations (e.g., */etc/*), and even writing books. We kept this edition of this book under revision control using Git while writing it; we used Subversion for the first edition.

Some of the useful features of revision control systems include:

- Making it very difficult to lose your work, especially when the repository is properly backed up.
- Facilitating change control practices, and encourage documenting why a change is being made.
- Allowing people in multiple locations to work together on a project, and to keep up with others' changes, without losing data by saving on top of each other or sending lots of unreadable emails.
- Allowing one person to work from multiple locations over time without losing work or stepping on changes made at other locations.

- Allowing you to back out changes easily or to see exactly what has changed between one revision and another (except binary files). If you follow effective logging practices, they will even tell you why a change was made.

Systems like CVS and Subversion also allow a form of keyword expansion that lets you embed revision metadata in nonbinary files.

There are many different free and commercial revision control systems, and if you are reading this book you should be using one! If you already know one, just use that. If your company has a standard tool, use that one. If neither of those help you choose, then use Git, Bazaar, or Mercurial. Do not use Subversion, CVS, RCS, or any of the older systems unless you have no choice. We'll briefly cover pros, cons, and basic usage for Git, Bazaar, Mercurial, and Subversion in this appendix, all of which either come with or are available for every major modern operating system. But before that we need to give a bit of background.

First, all the modern revision control systems are *distributed*, while older ones like Subversion and CVS are *centralized*. This is a major and fundamental difference, with some significant implications. In the older centralized systems, there is a central server, as the name implies, often maintained and backed up by your IT department, which is good. To do most useful things you need to connect to that server, which can be bad since that's much slower than local disk access and may not be feasible, while traveling, for example. Also, in those systems you can check out only part of the repository ("repo"), and thus you often have one large repo for the entire company, and you just check out and work on the parts you need. These systems also do the keyword expansion we mentioned; we'll show that in the section on Subversion. Finally, to *commit* is also to *publish*, which may be considered either a feature or a bug in such systems, but is probably more likely at least undesirable, if not quite a bug.

The distributed systems, on the other hand, do not have a central server, though often one copy is designated as the "source of truth" by convention. The repo you are working on is a complete copy, and it's just as good as anyone else's. That's a major change from the ability to just check out part of a repo. These systems do not do any keyword expansion and a commit is not the same as a publish, which requires an additional *push* step and a network connection to the remote repo. But they're local for all but *push/pull* operations and thus really fast.



With CVS or Subversion, you don't have to think about backups. If the code is committed, it's someplace else and probably backed up by IT (commit == publish). That is often *not* true with the modern distributed systems. They have local repos, so committed code stays local (commit != publish) until and unless you *push* (publish) it to a remote repo. If you never *push* you have only one local copy, so make sure you have good backups! A great tool for that is etc-keeper, discussed later in this appendix. The repository is inside */etc/*, so if you accidentally `rm -rf /etc` there goes the repo too.

The second major point is that Git unquestionably won the war, and “everyone” uses it, everywhere. OK, not quite everyone, since if you are still reading this you probably don't. But an awful lot of people do use it, and arguably a large part of the reason is [GitHub](#). So why are we not jumping fully on that bandwagon? We're glad you asked.

If you are a full-time developer working on a large<sup>1</sup> project, you're using Git already, and it's awesome. But if you are a more casual user, say a sysadmin with a collection of scripts, Git can be less awesome.<sup>2</sup> It is less actively user-hostile than it used to be, but it's still very complicated to use, and we have seen no good mental model for how it works. Far too often, you have to really understand Git's guts in order to use it for anything nontrivial, and that's just ugly. Git is also made out of razor blades and chainsaws—blazingly fast, extremely powerful, but dangerous; you can hurt yourself with it. Git history, for example, is very malleable, and it considers this a feature, not a bug. It uses hashes and dates instead of human-readable revision numbers, and though there are good reasons for this it can be quite inconvenient. Finally, the Git “index” is different; none of the other common tools have this, but it does allow for a really handy trick where you can make stream-of-consciousness changes but later commit them in logical blocks using `git add -p` or `git commit -p`. We think that it's a very powerful tool that's not suitable for beginners or casual users. But...it's everywhere and used by everyone, and that's also a powerful argument.

If you are interested in the history of revision control, see “[Understanding Version-Control Systems](#)” by Eric Raymond for a lot of detail. To see an amazing example and just a really cool thing, check out the [Unix History Repository](#).

If you are going to start using revision control just by yourself, go jump in. But if you are going to start using it in a team, you must first decide:

---

<sup>1</sup> We should say “web-scale,” to be buzzword-compliant.

<sup>2</sup> See also “[10 Things I Hate About Git](#)” by Steve Bennett.

- Which system or product to use
- The update, commit, tag, and branch policies
- The location of the central (and well-backed-up!) repository, if applicable
- The structure of the project or directories in the repository, if applicable

This appendix is enough to get you started individually, but it barely scratches the surface; see *Version Control with Git*, 2nd Edition, by Jon Loeliger and Matthew McCullough or *Version Control with Subversion*, 2nd Edition, by C. Michael Pilato, Ben Collins-Sussman, and Brian Fitzpatrick, both from O'Reilly, for more in-depth introductions to revision control and complete details on the respective systems. Both have excellent treatments of the general concepts, although the Subversion book covers repository structure in more detail due to its potentially multiproject nature. Both also cover revision control policy. If your company has change control or related policies, use them. If not, we recommend you commit and update early and often. If you are working as a team, we strongly recommend reading some of the books listed in this appendix and carefully planning out a strategy. It will save vast amounts of time in the long run.

## See Also

- “*Understanding Version-Control Systems*” by Eric Raymond
- *Unix History Repository*
- “*A Visual Guide to Version Control*” on BetterExplained
- *Backup & Recovery* by W. Curtis Preston (O'Reilly)
- *reposurgeon*, a tool for converting from one system to another

## Git

Git is the de facto leader in revision control and is probably used by more projects and more people than all the other systems combined. But if you choose to use it, be prepared to use Google. A lot.

Git was originally written by Linus Torvalds for the Linux kernel project after the vendor of the previous system changed the licensing, but he very quickly turned it over to others. The design is heavily influenced by Torvalds's years of experience on that massive and globally distributed project, and it is written by hardcore programmers for hardcore programmers. It is extremely powerful and flexible, but often quite complicated. The learning curve is unquestionably worth it for dedicated developers, but more casual or intermittent users may struggle.

## Pros

- Extremely popular and used everywhere.
- Extremely fast, powerful, and flexible.
- `git add -p` and `git commit -p` account for how code is really written.
- Has <https://github.com/>, <https://gitlab.com/>, etc.
- History is very malleable.

## Cons

- You can perform operations that can cause data loss!
- Harder to understand and use for more than very basic tasks than other tools.
- Inconsistent and complex command-line use.
- History is very malleable.
- Uses hashes and dates instead of human-readable revision numbers.

## Example

This example is not suitable for enterprise or multiuser access (see the “See Also” section for links to more information). This is just the basics, but it will get you started and you can ramp up from here if you need to.

If Git is not already installed, you should install it using the preferred package manager for your operating system.

The `git` command (with no options), `git help`, and `git help command` all give you helpful hints and reminders.

Configure Git on your machine (see `~/.gitconfig`, and the `.git/config` that the `init` command will create):

```
/home/jp$ git config --global user.name "JP Vossen"  
/home/jp$ git config --global user.email "jp@jpsdomain.org"  
/home/jp$ git config --global core.pager "less -R"  
/home/jp$ git config --global color.ui true
```



If you do not set your name and email as shown here, you will probably get a message complaining about that. The message should be pretty clear about what to do.

You might also consider:

```

/home/jp$ git config --global alias.co checkout
/home/jp$ git config --global alias.br branch
/home/jp$ git config --global alias.ci commit
/home/jp$ git config --global alias.st status
/home/jp$ git config --global alias.last 'log -1 HEAD'

```

Create a new repository for personal use in a home directory:

```

/home/jp$ git init myrepo
Initialized empty Git repository in /home/jp/myrepo/.git/

```

Create a new script and commit it:

```

/home/jp$ cd myrepo

/home/jp/myrepo$ cat << EOF > hello
> #!/bin/bash -
> echo 'Hello World!'
> EOF

/home/jp/myrepo$ chmod +x hello

/home/jp/myrepo$ git add hello

/home/jp/myrepo$ git commit -m 'Initial import of shell script'
[master (root-commit) 62cb49e] Initial import of shell script
1 file changed, 2 insertions(+)
create mode 100755 hello

```



`git add` is *not* the same as *add* in other tools! Once you add a file in the other tools, changes to that file are always committed. In Git, *add* means “add the changes I just made to the index.” So if you make a change, and add it, then make another change, that second change is *not* in the index and will not be committed unless you add the file again, or commit using `-a`. This sounds very annoying, and it is for basic use, but it’s part of the whole “index” concept that makes some other neat things possible, as we’ll see.



If you do not use the `-m message` option an editor will pop up and you can create a commit log in that. Which editor will appear and how you change that will depend on your OS and distribution; consult the appropriate documentation if you wish to change the editor.

Check the status of your sandbox:

```

/home/jp/myrepo$ git status
On branch master
nothing to commit, working directory clean

```

Add a new script to revision control:



```
/home/jp/scripts$ cat << EOF > mcd
> #!/bin/bash -
> mkdir -p "$1"
> cd "$1"
> EOF
```

```
/home/jp/myrepo$ chmod +x mcd
```

```
/home/jp/myrepo$ git status
```

On branch master

Untracked files:

(use "git add <file>..." to include in what will be committed)

mcd

nothing added to commit but untracked files present (use "git add" to track)

```
/home/jp/myrepo$ git add mcd
```

```
/home/jp/myrepo$ git status
```

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: mcd

```
/home/jp/myrepo$ git commit -m 'Added new script: mcd'
```

[master a2c254d] Added new script: mcd

1 file changed, 3 insertions(+)

create mode 100755 mcd

Make a change, then check the difference:

```
/home/jp/myrepo$ vi hello
```

```
/home/jp/myrepo$ git diff
```

```
1 diff --git a/hello b/hello
2 index 353223d..f36eea4 100644
3 --- a/hello
4 +++ b/hello
5 @@ -1,2 +1,2 @@
6  #!/bin/bash -
7  -echo 'Hello World!'
8  +echo 'Hello Mom!'
```

```
/home/jp/myrepo$ git status
```

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: hello

no changes added to commit (use "git add" and/or "git commit -a")



If you get a bunch of garbage escape characters on the screen when you run `git diff`, try setting `git config --global core.pager "less -R"`.

Commit the change using `-a` and thus avoiding `git add hello`:

```
/home/jp/myrepo$ git commit -a -m 'Fine tuning'
[master e1f0b2f] Fine tuning
1 file changed, 1 insertion(+), 1 deletion(-)
```

See the history of the repository or just one file:

```
/home/jp/myrepo$ git log
1 commit e1f0b2f8e5c489d8c9112014cf494773712786b0
2 Author: JP Vossen <jp@jpsdomain.org>
3 Date: Sun Jul 3 22:56:38 2016 -0400
4
5 Fine tuning
6
7 commit a2c254d61e95eb4719746f196b66019446061d51
8 Author: JP Vossen <jp@jpsdomain.org>
9 Date: Sun Jul 3 22:52:36 2016 -0400
10
11 Added new script: mcd
12
13 commit 62cb49ee962d929122051c421128fea95d571ebb
14 Author: JP Vossen <jp@drake.jpsdomain.org>
15 Date: Sun Jul 3 22:44:15 2016 -0400
16
17 Initial import of shell script
```

```
/home/jp/myrepo$ git log hello
1 commit e1f0b2f8e5c489d8c9112014cf494773712786b0
2 Author: JP Vossen <jp@jpsdomain.org>
3 Date: Sun Jul 3 22:56:38 2016 -0400
4
5 Fine tuning
6
7 commit 62cb49ee962d929122051c421128fea95d571ebb
8 Author: JP Vossen <jp@drake.jpsdomain.org>
9 Date: Sun Jul 3 22:44:15 2016 -0400
10
11 Initial import of shell script
```

Revert to the older version after all. There are other ways to do this, depending on what other changes you may have in your working directory, but this is simple if not intuitive:

```
/home/jp/myrepo$ git checkout 62cb49ee962d929122051c421128fea95d571ebb hello
```

```
/home/jp/myrepo$ cat hello
#!/bin/bash -
echo 'Hello World!'
```

```
/home/jp/myrepo$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   hello
```

```
/home/jp/myrepo$ git diff
```

But wait! We made a change, and *status* sees it but *diff* does not. Why? Because it already did a `git add`, so the change is *staged* or *cached*:

```
/home/jp/myrepo$ git diff --cached
1 diff --git a/hello b/hello
2 index f36eea4..353223d 100755
3 --- a/hello
4 +++ b/hello
5 @@ -1,2 +1,2 @@
6  #!/bin/bash -
7  -echo 'Hello Mom!'
8  +echo 'Hello World!'
```

We warned you...

## See Also

- `man git`
- `git help`
- <https://github.com/features>
- <https://about.gitlab.com/>
- [https://en.wikipedia.org/wiki/Git\\_\(software\)](https://en.wikipedia.org/wiki/Git_(software))
- <https://git-scm.com/>
- <http://xkcd.com/1597/>
- *Pro Git*, 2nd Edition, by Scott Chacon and Ben Straub (Apress)
- *Version Control with Git*, 2nd Edition, by Jon Loeliger and Matthew McCullough (O'Reilly)
- “10 Things I Hate About Git” by Steve Bennett
- “Aha! Moments When Learning Git” on BetterExplained
- The [EasyGit](#) wrapper

- Recipe 16.16

## Bazaar

Bazaar was Canonical's answer to Git, but it lost the war and is basically in maintenance mode.

### Pros

- Not Git.
- Extremely user-friendly with awesome docs.
- Cross-platform (Python) with several GUI tools: QBzr (Qt), Loggerhead (web), and others.
- Uses incrementing integer revision numbers.
- History is immutable.
- Has [Launchpad](#).

### Cons

- Not Git.
- Lost the war and is not-quite-dead.
- Not as fast as Git, but that almost never matters.
- Not nearly as well known as Git.

## Example

This example is not suitable for enterprise or multiuser access (see the “See Also” section for links to more information). This is just to show how easy the basics are.

If Bazaar is not already installed, you should install it using the preferred package manager for your operating system.

The *bzr* command (with no options), *bzr help*, and *bzr help command* all give you helpful hints and reminders.

Create a new repository for personal use in a home directory:

```
/home/jp$ bzr init myrepo
Created a standalone tree (format: 2a)
```

Create a new script and commit it:

```
/home/jp$ cd myrepo
```

```
/home/jp/myrepo$ cat << EOF > hello
> #!/bin/bash -
> echo 'Hello World!'
> EOF
```

```
/home/jp/myrepo$ chmod +x hello
```

```
/home/jp/myrepo$ bzip add hello
adding hello
```

```
/home/jp/myrepo$ bzip commit -m 'Initial import of shell script'
Committing to: /home/jp/myrepo/
added hello
Committed revision 1.
```



If you do not use the `-m message` option an editor will pop up and you can create a commit log in that. Which editor will appear and how you change that will depend on your OS and distribution; consult the appropriate documentation if you wish to change the editor.

Check the status of your sandbox:

```
/home/jp/myrepo$ bzip status
```

Add a new script to revision control:

```
/home/jp/scripts$ cat << EOF > mcd
> #!/bin/bash -
> mkdir -p "$1"
> cd "$1"
> EOF
```

```
/home/jp/myrepo$ chmod +x mcd
```

```
/home/jp/myrepo$ bzip status
unknown:
mcd
```

```
/home/jp/myrepo$ bzip add mcd
adding mcd
```

```
/home/jp/myrepo$ bzip status
added:
mcd
```

```
/home/jp/myrepo$ bzip commit -m 'Added new script: mcd'
Committing to: /home/jp/myrepo/
added mcd
Committed revision 2.
```

Make a change, then check the difference:

```
/home/jp/myrepo$ vi hello
```

```
/home/jp/myrepo$ bzip diff
=== modified file 'hello'
--- hello      2016-07-04 03:26:32 +0000
+++ hello      2016-07-04 03:28:11 +0000
@@ -1,2 +1,2 @@
     #!/bin/bash -
     -echo 'Hello World!'
     +echo 'Hello Mom!'
```

```
/home/jp/myrepo$ bzip status
modified:
    hello
```

Commit the change:

```
/home/jp/myrepo$ bzip commit -m 'Fine tuning'
Committing to: /home/jp/myrepo/
modified hello
Committed revision 3.
```

See the history of the repository or just one file:

```
/home/jp/myrepo$ bzip log
-----
revno: 3
committer: JP Vossen <jp@ringo.jpsdomain.org>
branch nick: myrepo
timestamp: Sun 2016-07-03 23:28:48 -0400
message:
    Fine tuning
-----

revno: 2
committer: JP Vossen <jp@ringo.jpsdomain.org>
branch nick: myrepo
timestamp: Sun 2016-07-03 23:27:50 -0400
message:
    Added new script: mcd
-----

revno: 1
committer: JP Vossen <jp@ringo.jpsdomain.org>
branch nick: myrepo
timestamp: Sun 2016-07-03 23:26:32 -0400
message:
    Initial import of shell script

/home/jp/myrepo$ bzip log hello
-----
revno: 3
committer: JP Vossen <jp@ringo.jpsdomain.org>
branch nick: myrepo
timestamp: Sun 2016-07-03 23:28:48 -0400
message:
```

```
Fine tuning
-----
revno: 1
committer: JP Vossen <jp@ringo.jpdomain.org>
branch nick: myrepo
timestamp: Sun 2016-07-03 23:26:32 -0400
message:
    Initial import of shell script
```

Revert to the older version after all:

```
/home/jp/myrepo$ bzz revert -r1 hello
M hello

/home/jp/myrepo$ bzz status
modified:
    hello

/home/jp/myrepo$ bzz diff
=== modified file 'hello'
--- hello      2016-07-04 03:28:48 +0000
+++ hello      2016-07-04 03:29:44 +0000
@@ -1,2 +1,2 @@
    #!/bin/bash -
    -echo 'Hello Mom!'
    +echo 'Hello World!'
```

## See Also

- `man bzz`
- `bzz help`
- [https://en.wikipedia.org/wiki/Bazaar\\_\(software\)](https://en.wikipedia.org/wiki/Bazaar_(software))
- <http://wiki.bazaar.canonical.com/Documentation>
- <http://wiki.bazaar.canonical.com/Workflows>
- *Bazaar Version Control* by Janos Gyerik (Packt)
- Recipe 16.16

## Mercurial

Mercurial was started at the same time as Git for the same reason, but never caught on quite as much.

## Pros

- Not Git.
- Extremely user-friendly with good docs.

- Cross-platform (Python) with several GUI tools.
  - Built-in web server (`hg serve` then <http://localhost:8000/>).
- Uses incrementing integer revision numbers + a hex ID.
  - The hex ID is unique and consistent across all repo clones, the integer isn't.
- History is immutable.
- Has Atlassian <https://bitbucket.org/>.

## Cons

- Not Git.
- Lost to Git but more active than Bazaar.
- Not as well known as Git.
- Not as fast as Git, but that almost never matters.

## Example

This example is not suitable for enterprise or multiuser access (see the “See Also” section for links to more information). This is just to show how easy the basics are.

If Mercurial is not already installed, you should install it using the preferred package manager for your operating system.

`hg` command (with no options), `hg help`, and `hg help command` all give you helpful hints and reminders..

Create a new repository for personal use in a home directory:

```
/home/jp$ hg init myrepo
```

Create a new script and commit it:

```
/home/jp$ cd /myrepo
```

```
/home/jp/myrepo$ cat << EOF > hello
> #!/bin/bash -
> echo 'Hello World!'
> EOF
```

```
/home/jp/myrepo$ chmod +x hello
```

```
/home/jp/myrepo$ hg add hello
```

```
/home/jp/myrepo$ hg commit -m 'Initial import of shell script'
```





If you do not use the `-m message` option an editor will pop up and you can create a commit log in that. Which editor will appear and how you change that will depend on your OS and distribution; consult the appropriate documentation if you wish to change the editor.

Check the status of your sandbox:

```
/home/jp/myrepo$ hg status
```

Add a new script to revision control:

```
/home/jp/scripts$ cat << EOF > mcd
> #!/bin/bash -
> mkdir -p "$1"
> cd "$1"
> EOF
```

```
/home/jp/myrepo$ chmod +x mcd
```

```
/home/jp/myrepo$ hg status
? mcd
```

```
/home/jp/myrepo$ hg add mcd
```

```
/home/jp/myrepo$ hg status
A mcd
```

```
/home/jp/myrepo$ hg commit -m 'Added new script: mcd'
```

Make a change, then check the difference:

```
/home/jp/myrepo$ vi hello
```

```
/home/jp/myrepo$ hg diff
diff -r 663ba0ec20f5 hello
--- a/hello      Sun Jul 03 23:38:54 2016 -0400
+++ b/hello      Sun Jul 03 23:39:15 2016 -0400
@@ -1,2 +1,2 @@
-#!/bin/bash -
-echo 'Hello World!'
+echo 'Hello Mom!'
```

```
/home/jp/myrepo$ hg status
M hello
```

Commit the change:

```
/home/jp/myrepo$ hg commit -m 'Fine tuning'
```

See the history of the repository or just one file:

```
/home/jp/myrepo$ hg log
changeset: 2:c88ab0cbfcda
```

```
tag:      tip
user:     JP Vossen <jp@jpsdomain.org>
date:     Sun Jul 03 23:39:38 2016 -0400
summary:  Fine tuning
```

```
changeset: 1:663ba0ec20f5
user:      JP Vossen <jp@jpsdomain.org>
date:     Sun Jul 03 23:38:54 2016 -0400
summary:  Added new script: mcd
```

```
changeset: 0:38ab693c1c72
user:      JP Vossen <jp@jpsdomain.org>
date:     Sun Jul 03 23:38:03 2016 -0400
summary:  Initial import of shell script
```

```
/home/jp/myrepo$ hg log hello
changeset: 2:c88ab0cbfcda
tag:      tip
user:     JP Vossen <jp@jpsdomain.org>
date:     Sun Jul 03 23:39:38 2016 -0400
summary:  Fine tuning
```

```
changeset: 0:38ab693c1c72
user:      JP Vossen <jp@jpsdomain.org>
date:     Sun Jul 03 23:38:03 2016 -0400
summary:  Initial import of shell script
```

Revert to the older version after all:

```
/home/jp/myrepo$ hg revert -r 1 hello
```

```
/home/jp/myrepo$ cat hello
#!/bin/bash -
echo 'Hello World!'
```

```
/home/jp/myrepo$ hg status
M hello
```

## See Also

- `man hg`
- `hg help`
- <https://en.wikipedia.org/wiki/Mercurial>
- <https://www.mercurial-scm.org/>
- <https://www.mercurial-scm.org/guide>
- Book: <http://hgbook.red-bean.com/>
- <https://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/>
- Recipe 16.16

# Subversion

According to the Subversion web site, “The goal of the Subversion project is to build a *version control system* that is a compelling replacement for CVS in the open source community.” Enough said.

## Pros

- Not Git.
- Newer than CVS and RCS.
- Simpler and arguably easier to understand and use than CVS (less historical baggage).
- Atomic commits means the commit either fails or succeeds as a whole, and makes it easy to track the state of an entire project as a single revision.
- Easy to access remote repositories.
- Allows easy renaming of files and directories while retaining history.
- Easily handles binary files (no native diff support) and other objects such as symbolic links.
- Central repository hacking is more officially supported, but less trivial.

## Cons

- Not Git.
- Older technology, revision control has moved to the distributed model.
- Can be complicated to build or install from scratch due to many dependencies. Use the version that came with your operating system if possible.



SVN tracks revisions by repository, which means that each commit has its own internal SVN revision number. Thus consecutive commits by a single person may not have consecutive revision numbers since the global repository revision is incremented as other changes (possibly to other projects) are committed by other people.

## Example

This example is not suitable for enterprise or multiuser access (see the “See Also” section for links to more information). This is just to show how easy the basics are. This example also has the `EDITOR` environment variable set to `nano` (export `EDI`

TOR='nano --smooth --const --nowrap --suspend'), which some people find more user-friendly than the default *vi*.

The `svn help` and `svn help help` commands are very useful.

Create a new repository for personal use in a home directory:

```
/home/jp$ svnadmin --fs-type=fsfs create /home/jp/svnroot
```

Create a new project and import it:

```
/home/jp$ cd /tmp

/tmp$ mkdir -p -m 0700 scripts/trunk scripts/tags scripts/branches

/tmp$ cd scripts/trunk

/tmp/scripts/trunk$ cat << EOF > hello
> #!/bin/sh
> echo 'Hello World!'
> EOF

/tmp/scripts/trunk$ cd ..

/tmp/scripts$ svn import /tmp/scripts file:///home/jp/svnroot/scripts

GNU nano 1.2.4                               File: svn-commit.tmp
```

```
Initial import of shell scripts
--This line, and those below, will be ignored--
```

```
A  .
                                     [ Wrote 4 lines ]
```

```
Adding      /tmp/scripts/trunk
Adding      /tmp/scripts/trunk/hello
Adding      /tmp/scripts/branches
Adding      /tmp/scripts/tags
```

```
Committed revision 1.
```

Check out the project and update it:

```
/tmp/scripts$ cd

/home/jp$ svn checkout file:///home/jp/svnroot/scripts
A  scripts/trunk
A  scripts/trunk/hello
A  scripts/branches
A  scripts/tags
Checked out revision 1.

/home/jp$ cd scripts
```

```

/home/jp/scripts$ ls -l
total 12K
drwxr-xr-x  3 jp jp 4.0K Jul 20 01:12 branches/
drwxr-xr-x  3 jp jp 4.0K Jul 20 01:12 tags/
drwxr-xr-x  3 jp jp 4.0K Jul 20 01:12 trunk/

/home/jp/scripts$ cd trunk/

/home/jp/scripts/trunk$ ls -l
total 4.0K
-rw-r--r--  1 jp jp 30 Jul 20 01:12 hello

/home/jp/scripts/trunk$ echo "Hi Mom..." >> hello

```

Check the status of your sandbox. Note how the `svn status` command is similar to our `cvs -qn update` hack in the “CVS” section earlier in this appendix:

```

/home/jp/scripts/trunk$ svn info
Path: .
URL: file:///home/jp/svnroot/scripts/trunk
Repository UUID: 29eeb329-fc18-0410-967e-b075d748cc20
Revision: 1
Node Kind: directory
Schedule: normal
Last Changed Author: jp
Last Changed Rev: 1
Last Changed Date: 2006-07-20 01:04:56 -0400 (Thu, 20 Jul 2006)

/home/jp/scripts/trunk$ svn status -v
      1      1 jp      .
M      1      1 jp      hello

/home/jp/scripts/trunk$ svn status
M      hello

/home/jp/scripts/trunk$ svn update
At revision 1.

```

Add a new script to revision control:

```

/home/jp/scripts/trunk$ cat << EOF > mcd
> #!/bin/sh
> mkdir -p "$1"
> cd "$1"
> EOF

/home/jp/scripts/trunk$ svn st
?      mcd
M      hello
/home/jp/scripts/trunk$ svn add mcd
A      mcd

```

Commit changes:

```

/home/jp/scripts/trunk$ svn ci

GNU nano 1.2.4                               File: svn-commit.tmp* Tweaked hello
* Added mcd
--This line, and those below, will be ignored--

M    trunk/hello

A    trunk/mcd

[ Wrote 6 lines ]

Sending      trunk/hello
Adding      trunk/mcd
Transmitting file data ..
Committed revision 2.

```

Update the sandbox, make another change, then check the difference:

```

/home/jp/scripts/trunk$ svn up
At revision 2.

/home/jp/scripts/trunk$ vi hello

/home/jp/scripts/trunk$ svn diff hello
Index: hello
=====
--- hello      (revision 2)
+++ hello      (working copy)
@@ -1,3 +1,3 @@
 #!/bin/sh
 echo 'Hello World!'
-Hi Mom...
+echo 'Hi Mom...'

```

Commit the change, avoiding the editor by putting the log entry on the command line:

```

/home/jp/scripts/trunk$ svn -m 'Fine tuning' commit
Sending      trunk/hello
Transmitting file data .
Committed revision 3.

```

See the history of the file:

```

/home/jp/scripts/trunk$ svn log hello
-----
r3 | jp | 2006-07-20 01:23:35 -0400 (Thu, 20 Jul 2006) | 1 lineFine tuning
-----
r2 | jp | 2006-07-20 01:20:09 -0400 (Thu, 20 Jul 2006) | 3 lines
* Tweaked hello
* Added mcd
-----

```

r1 | jp | 2006-07-20 01:04:56 -0400 (Thu, 20 Jul 2006) | 2 lines

Initial import of shell scripts

Add some revision metadata, and tell the system to expand it. Commit it and examine the change:

```
/home/jp/scripts/trunk$ vi hello

/home/jp/scripts/trunk$ cat hello
#!/bin/sh
# $Id$
echo 'Hello World!'
echo 'Hi Mom...'

/home/jp/scripts/trunk$ svn propset svn:keywords "Id" hello
property 'svn:keywords' set on 'hello'

/home/jp/scripts/trunk$ svn ci -m'* Added ID keyword' hello
Sending          hello

Committed revision 4.
```

```
/home/jp/scripts/trunk$ cat hello
#!/bin/sh
# $Id: hello 5 2006-07-21 09:09:34Z jp $</code></strong>
echo 'Hello World!'
echo 'Hi Mom...'
```

Compare the current revision to r2, revert to that older (broken) revision, realize we goofed and get the most recent revision back:

```
/home/jp/scripts/trunk$ svn diff -r2 hello
Index: hello
=====
--- hello      (revision 2)
+++ hello      (working copy)
@@ -1,3 +1,4 @@
  #!/bin/sh
+# $Id$
  echo 'Hello World!'
  -Hi Mom...
+echo 'Hi Mom...'

Property changes on: hello
-----
Name: svn:keywords
+ Id

/home/jp/scripts/trunk$ svn update -r2 hello
UU hello
Updated to revision 2.
```

```
/home/jp/scripts/trunk$ cat hello
#!/bin/sh
echo 'Hello World!'
Hi Mom...

/home/jp/scripts/trunk$ svn update -rHEAD hello
UU hello
Updated to revision 4.

/home/jp/scripts/trunk$ cat hello
#!/bin/sh
# $Id: hello 5 2006-07-21 09:09:34Z jp $
echo 'Hello World!'
echo 'Hi Mom...'
```

## See Also

- `man svn`
- `man svnadmin`
- `man svndumpfilter`
- `man svnlook`
- `man svnserve`
- `man svnversion`
- [The Subversion website](#)
- [TortoiseSVN](#), a simple SVN frontend for Explorer (cool!)
- *[Version Control with Subversion](#)* by C. Michael Pilato, Ben Collins-Sussman, and Brian Fitzpatrick
  - [“Appendix B: Subversion for CVS Users”](#)
- [The FreeBSD guide to using Subversion](#)
- [SVN static builds](#) for Solaris, Linux, and macOS
- Better SCM Initiative [version control system comparison](#)
- [“A Visual Guide to Version Control”](#) on BetterExplained
- [Recipe 16.16](#)

## Meld

Meld is not a revision control tool itself; it is a very useful graphical diff and merge tool that can work with revision control systems. When run normally, it allows you to compare and merge files and directories. When run from a revision control sandbox,



it will compare the working copy to the version under revision control and show you what you've changed. Trust us, it's awesome.

## Pros

- Cross-platform (Python)
- Available for all or most Linux distributions
- Windows installer
- Unofficial Mac installers

## Cons

- None

## Example

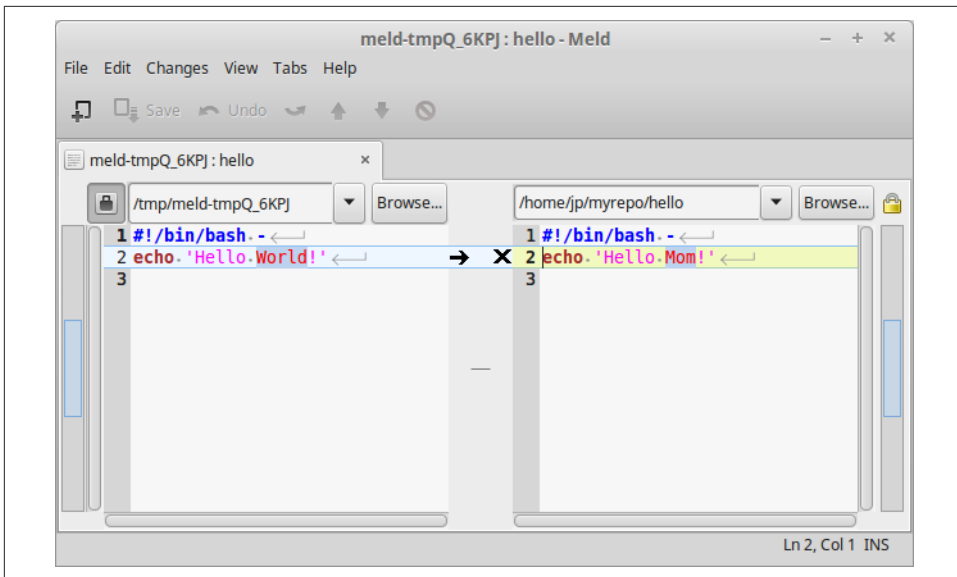


Figure D-1. Meld in action

## See Also

- `man meld`
- <http://meldmerge.org/>
- [https://en.wikipedia.org/wiki/Meld\\_\(software\)](https://en.wikipedia.org/wiki/Meld_(software))

# etckeeper

etckeeper is not a revision control tool itself, but it uses one to put your */etc/* directory under revision control. It's available in all or most Linux distributions, and it hooks into *cron* to do daily commits and the package manager to do commits before and after package operations. It also works around the issues of files appearing and disappearing, ownership, permissions and such that revision control systems usually don't handle all by themselves. It uses the underlying tool's "ignore" file to ignore files that change too often or are otherwise not useful to revision.

Out of the box, Meld creates a repository in */etc/* and starts committing. You can configure the underlying revision control system to push to a remote repository as a backup as well.

Which revision control system it uses varies by distribution, and it's configurable as well.

Here are a few tips, if you're thinking about using etckeeper:

- There are security implications to storing the */etc/shadow* file in etckeeper. See the *README* for details.
- You will need to install the Extra Packages for Enterprise Linux (EPEL) repository for Red Hat Enterprise, CentOS, and similar RPM distros.
- etckeeper will not initialize or commit for you, like Debian does. After installing the RPM, you will need to run `sudo etckeeper init` and `sudo etckeeper commit First commit` before it will start working for you.

## Pros

- Set-it-and-forget-it revision control for */etc/*!

## Cons

- See the potential security implication.
- The out-of-the-box configuration is local only.

## Example

Here's an example install on a mostly stock Debian (Jessie) system:

```
[jp@jessie:T0:L1:C19:J0:2016-07-04_15:47:25_EDT]
/home/jp$ sudo apt-get update
[sudo] password for jp:
...
```

```

Fetched 7,652 B in 4s (1,796 B/s)
Reading package lists... Done

[jp@jessie:T0:L1:C20:J0:2016-07-04_15:47:50_EDT]
/home/jp$ sudo apt-get install etckeeper
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  git git-man liberror-perl
Suggested packages:
  git-daemon-run git-daemon-sysvinit git-doc git-el git-email git-gui gitk gitweb
  git-arch git-cvs
  git-mediawiki git-svn
The following NEW packages will be installed:
  etckeeper git git-man liberror-perl
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 4,587 kB of archives.
After this operation, 23.7 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
...
Setting up etckeeper (1.15) ...
Initialized empty Git repository in /etc/.git/
[master (root-commit) 6d597ca] Initial commit
Author: jp <jp@jessie.jpsdomain.org>
1324 files changed, 32995 insertions(+)
create mode 100755 .etckeeper
create mode 100644 .gitignore
...
create mode 100644 xml/catalog
create mode 100644 xml/docutils-common.xml
create mode 100644 xml/xml-core.xml

/home/jp$ cd /etc

/etc$ sudo git status
On branch master
nothing to commit, working directory clean

/etc$ cat /etc/cron.daily/etckeeper
#!/bin/sh
set -e
if [ -x /usr/bin/etckeeper ] && [ -e /etc/etckeeper/etckeeper.conf ]; then
    . /etc/etckeeper/etckeeper.conf
    if [ "$AVOID_DAILY_AUTOCOMMITS" != "1" ]; then
        # avoid autocommit if an install run is in progress
        lockfile=/var/cache/etckeeper/packagelist.pre-install
        if [ -e "$lockfile" ] && [ -n "$(find "$lockfile" -mtime +1)" ]
        then
            rm -f "$lockfile" # stale

```

```

        fi
        if [ ! -e "$lockfile" ]; then
            AVOID_SPECIAL_FILE_WARNING=1
            export AVOID_SPECIAL_FILE_WARNING
            if etckeeper unclean; then
                etckeeper commit "daily autocommit" >/dev/null
            fi
        fi
    fi
fi

```

Now *etckeeper* will commit daily and before and after package operations. But you can commit manually as well, and you can use all of the features of the underlying revision control system:

```
/etc$ sudo useradd carl
```

```
/etc$ sudo git status
```

```
On branch master
```

```
Changes not staged for commit:
```

```
(use "git add <file>..." to update what will be committed)
```

```
(use "git checkout -- <file>..." to discard changes in working directory)
```

```

modified:   group
modified:   group-
modified:   gshadow
modified:   gshadow-
modified:   passwd
modified:   shadow
modified:   subgid
modified:   subgid-
modified:   subuid
modified:   subuid-

```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

```
/etc$ sudo etckeeper commit 'Added a user for Carl'
```

```
[master 8b58601] Added a user for Carl
```

```
Author: jp <jp@jessie.jpsdomain.org>
```

```
11 files changed, 12 insertions(+), 2 deletions(-)
```

```
/etc$ sudo git status
```

```
On branch master
```

```
nothing to commit, working directory clean
```

## See Also

- *etckeeper*
- `man etckeeper`

- <http://etckeeper.branchable.com/>  
— <http://etckeeper.branchable.com/README/>

## Other

Finally, it is worth noting that some word processors, such as LibreOffice Writer and Microsoft Word, have three relevant features: document comparison, change tracking, and versions.

### Document Comparison

Document comparison allows you to compare documents when their native file format makes use of other *diff* tools difficult. You would use this when you have two copies of a document that didn't have change tracking turned on, or when you need to merge feedback from various sources.

While it is trivial to unzip the *content.xml* file from a given ODF file, the result has no line breaks and is not terribly pretty or readable. See Recipe 12.5 for a *bash* script that will do this low-level kind of difference.

Refer to [Table D-1](#) at the end of this section for information on how to access the built-in GUI comparison function, which is much easier than trying to do it manually.

### Change Tracking and Versions

The change-tracking feature saves information about changes made to a document. Review mode uses various copyediting markup on the screen to display who did what, when. This is obviously useful for all kinds of creation and editing purposes, but please read our warning.

The versions feature allows you to save more than one version of a document in a single file. This can be handy in all sorts of odd ways. For example, we've seen router configurations copied and pasted from a terminal into different versions inside the same document for archival and change control purposes.



The change-tracking and versions features will cause your document to continually grow in size, since items that are changed are still kept and deleted items are not really deleted, but only marked as deleted.

Also, if accidentally turned on, change tracking and versions can be very dangerous information leaks! For example, if you send similar proposals to competing companies after doing a search and replace and other editing, someone at one of those companies can see exactly what you changed and when you changed it. The most recent versions of these tools have various methods that attempt to warn you or clear private information before a given document is converted to PDF or emailed, but take a look at any word processor attachments you receive in email, especially from vendors. You may be surprised.

## Accessing These Features

**Table D-1** shows where to find the features described here in LibreOffice Writer and Microsoft Word.

*Table D-1. Word processor functions*

Feature	Writer menu option	Word menu option
Document comparisons	Edit → Compare Document	Tools → Compare and Merge Documents
Change tracking	Edit → Changes	Tools → Track Changes
Versions	File → Versions	File → Versions