# How do I discard unstaged changes in Git?

Asked 11 years, 2 months ago Active 2 months ago Viewed 2.4m times

4557

1384

How do I discard changes in my working copy that are not in the index?

git version-control

edited Apr 17 at 4:46
Venkat
2,161 2 13 42

asked Sep 9 '08 at 19:33
Readonly
272k 96 192 196

9 `git-clean` only removes untracked files from the working tree git-scm.com/docs/git-clean – Yega Sep 15 '16 at 12:29

22 To clarify Asenar's comment above, `git-clean -df` can be dangerous. It will delete local untracked files (e.g. covered by a .gitignore) Read all below carefully and consider git checkout . instead – jacanterbury Oct 7 '16 at 8:05

13 'git clean -df ' Be warned! I tried that and lost key folders that are unable to be restored... Ouch! – Gabe Karkanis Oct 27 '16 at 21:01

31 hitting `git status` gives a suggestion on how to do that! `git checkout -- .` – Paulo Dec 21 '17 at 10:42

## 35 Answers

1 2 next

2548

Another quicker way is:

```
git stash save --keep-index --include-untracked
```

After that, you can drop that stash with a `git stash drop` command if you like.

edited Sep 11 '18 at 0:14

Brandon Minnick
**8,164** 12 34 88

answered Sep 9 '08 at 19:39

Greg Hewgill
**727k** 159 1045 1190

118 And to be thorough about it, you'd want `--include-untracked` as well. – T.J. Crowder Mar 23 '15 at 7:45

9 @KarimSamir: The question specifically asks about changes that are *not in the index*. The `git reset` command will discard changes in the index too. – Greg Hewgill Apr 12 '15 at 17:28

134 git checkout -- . is much faster – Frank Apr 17 '15 at 16:16

37 Neither the `git stash` , nor any variety of `git checkout` will discard unstaged deletes. According to the output of `git status` , the actual correct answer here is some flavor `git reset HEAD` – Chris Warth May 27 '15 at 22:27

118 This pollutes the stash stack. `git checkout -- .` does the job with one command only. – Felipe Tonello Sep 9 '15 at 11:17

8

**2019 update:**

Since July 2019, there has been a new command which does exactly this: `git restore .`

In `git status` , now Git recommends using this command instead of `git checkout` as it used to.

While this command can also be used to restore the working tree to a specific commit or to restore the content of the index, by default, the working tree is restored to the state in the index (what is asked here).

So, in order to restore the files matching a pathspec (getting rid of their unstaged changes), you would do:

```
git restore <pathspec>
```

And to remove all unstaged changes in the current repository:

```
git restore .
```

```
git clean -dff
```

*Be very careful with this later command however as you might discard files you did not intend to get rid of.*

Note on `git restore` : as this is a new command, its man page gives a warning:

> This command is experimental. The behavior may change.

So it is possible that this answer might become outdated if the behaviour does change in the future. It might thus be wise to run a quick `man git-restore` before using it.

edited Sep 26 at 21:39

answered Sep 11 at 1:30

prosoitos

**1,903** 1 10 21

---

4

To do a permanent discard: `git reset --hard`

To save changes for later: `git stash`

edited Sep 19 at 20:42



siddhantsomani

**355** 4 8

answered Jul 3 at 12:41

SANGEETHA P.H.

**119** 1 3

---

6

you have a very simple git command `git checkout .`

answered Aug 27 at 7:58

khem raj regmi

**756** 7 13

---

5030

```
git checkout -- .
```

+300

For a specific file use:

```
git checkout -- path/to/file/to/revert
```

`--` here to remove argument ambiguation.

edited Nov 26 '18 at 23:54
mja
714 12 19

answered Sep 9 '08 at 19:37
Tobi
55.8k 5 28 36

112 This seems to be the git canonical way. i.e. exactly what git tells you to do if you type `git status` – ABMagil Aug 18 '14 at 16:01

26 Doesn't work if there are untracked files. Git says `error: The following untracked working tree files would be overwritten by checkout: ... .` – Michael Iles Aug 24 '14 at 13:26

87 newbie question, what does "git checkout -- ." mean semantically? – kaid Sep 7 '14 at 19:21

115 @Ninjack `git checkout -- .` means the same thing as `git checkout .` , except that you're explicit about the fact that you're not specifying the branch name. They both say checkout the HEAD version on the branch I am currently on for '.' or './'. If you do `git checkout branch-name directory-or-file-name` in general, you get the HEAD version of `directory-or-file-name` on branch `branch-name` . – akgill Oct 29 '14 at 19:55

22 IMO this variant is imperfect, as it doesn't handle situation when your changed repository is not on the HEAD revision at the moment of changes cleaning and you DO NOT want to update it to HEAD, and want to just clean the changes. – alexykot Jan 5 '15 at 17:27

54

If you merely wish **to remove changes to existing files**, use `checkout` (documented here).

```
git checkout -- .
```

- No branch is specified, so it checks out the current branch.
- The double-hyphen ( `--` ) tells Git that what follows should be taken as its second argument (path), that you skipped specification of a branch.
- The period ( `.` ) indicates all paths.

```
git clean -i
```

- The `-i` option initiates an interactive `clean`, to prevent mistaken deletions.
- A handful of other options are available for a quicker execution; see the documentation.

If you wish **to move changes to a holding space for later access**, use `stash` (documented here):

```
git stash
```

- All changes will be moved to Git's Stash, for possible later access.
- A handful of options are available for more nuanced stashing; see the documentation.

answered Mar 18 '18 at 0:19

jtheletter
**7,687** 4 37 45

---

89

Since no answer suggests the exact option combination that I use, here it is:

```
git clean -dfx
git checkout .
```

This is the online help text for the used `git clean` options:

```
-d
```

Remove untracked directories in addition to untracked files. If an untracked directory is managed by a different Git repository, it is not removed by default. Use `-f` option twice if you really want to remove such a directory.

```
-f
```

If the Git configuration variable `clean.requireForce` is not set to `false`, Git clean will refuse to delete files or directories unless given `-f`, `-n`, or `-i`. Git will refuse to delete directories within the `.git` subdirectory or file, unless a second `-f` is given.

Don't use the ignore rules from `.gitignore` (per directory) and `$GIT_DIR/info/exclude`, but do still use the ignore rules given with `-e` options. This allows removing all untracked files, including build products. This can be used (possibly in conjunction with `git reset`) to create a pristine working directory to test a clean build.

Also, `git checkout .` needs to be done in the root of the repo.

edited Mar 17 '18 at 23:50

jtheletter
**7,687** 4 37 45

answered Apr 28 '16 at 19:46

Martin G
**11.8k** 8 62 73

2 Also regarding the first command `git clean -dfx`, here is a tip I use to be on the safe side before running it: just run `git clean -d -x -n` before, to display the list of files-to-be-removed, then confirm the operation by running `git clean -d -x -f` (I put the argument `-n`, resp. `-f` in the end to be able to quickly change it in a terminal) – ErikMD Jan 15 '18 at 20:33

5 Quick note that this is unreversable, and if you have files in `.gitignore` you will lose them. So consider backing up your project before this. – Rob Apr 3 '18 at 3:47

11

No matter what state your repo is in you can always reset to any previous commit:

```
git reset --hard <commit hash>
```

This will discard all changes which were made after that commit.

edited Jan 29 '18 at 16:29

jdgregson
**971** 10 25

answered Feb 5 '16 at 0:59

msangel
**7,470** 3 38 60

1 This will also discard everything in the index (not just things not in the index), which is beyond what the OP is asking for. – Linus Arver Jul 8 '18 at 7:51

The easiest way to do this is by using this command:

```
git checkout -- .
```

https://git-scm.com/docs/git-checkout

In git command, stashing of untracked files is achieved by using:

```
git stash -u
```

http://git-scm.com/docs/git-stash

edited Oct 23 '17 at 9:30

answered Apr 12 '17 at 9:27

A H M Forhadul Islam
**1,127** 8 11

17 Twice I've come here, read this answer, and forgotten the `.` at the end. To future me: the period is *essential*! – bejado Jun 16 '17 at 17:13

2 I needed to get rid of all local changes in a sub directory, without blowing away every other change. This answer helped a lot, thanks – Ally Sep 7 '17 at 4:00

2 Please describe what the two commands do. It's really unhelpful to have no explanation. – Chris Kennedy Sep 9 '17 at 17:22

2 excellent. the checkout does in one command what the most popular one does in two. can also be followed up with `git clean -fd` to clean files not in the index. – oligofren Nov 28 '17 at 9:03

2 This is the only one that worked for me. – ScottyBlades Sep 28 '18 at 21:31

6

If it's almost impossible to rule out modifications of the files, have you considered ignoring them? If this statement is right and you wouldn't touch those files during your development, this command may be useful:

```
git update-index --assume-unchanged file_to_ignore
```

answered Jul 31 '17 at 15:57

Jesús Castro
**1,484** 1 17 22

6

```
git rm .gitattributes
git add -A
git reset --hard
```

edited Jul 18 '17 at 12:23
Vadim Kotov
**5,689** 7 37 49

answered Feb 8 '17 at 11:58
SDV
**174** 1 6

---

6

If you are in case of submodule and no other solutions work try:

- To check what is the problem (maybe a "dirty" case) use:

```
git diff
```

- To remove stash

```
git submodule update
```

edited Jul 18 '17 at 12:22
Vadim Kotov
**5,689** 7 37 49

answered Oct 1 '15 at 21:32

onalbi
**1,427** 17 30

---

7

You could create your own alias which describes how to do it in a descriptive way.

I use the next alias to discard changes.

### Discard changes in a (list of) file(s) in working tree

```
discard = checkout --
```

Then you can use it as next to discard all changes:

Or just a file:

```
discard filename
```

Otherwise, if you want to discard all changes and also the untracked files, I use a mix of checkout and clean:

### Clean and discard changes and untracked files in working tree

```
cleanout = !git clean -df && git checkout -- .
```

So the use is simple as next:

```
cleanout
```

Now is available in the next Github repo which contains a lot of aliases:

- https://github.com/GitAlias/gitalias

edited Jun 5 '17 at 4:51

answered Jun 5 '17 at 4:44

Pau

**7,511** 7 39 69

---

1824

It seems like the complete solution is:

```
git clean -df
git checkout -- .
```

`git clean` removes all untracked files (**warning**: while it won't delete ignored files mentioned directly in .gitignore, **it may delete ignored files residing in folders**) and `git checkout` clears all unstaged changes.

edited Mar 29 '17 at 2:32

answered Aug 29 '12 at 18:28

113 The other two answers don't actually work, this one did. – John Hunt Sep 1 '14 at 12:23

18 @dval this is becues the first command removed the unindexed files and the second one removed the unstaged changes (of indexed files). So if you did not have any staged changes this it is the same as reverting to the last commit with `git reset --hard` – Amanuel Nega Oct 31 '14 at 10:47

3 use **-dff** if the untracked directory is a git clone. – accuya Dec 16 '14 at 2:52

86 Be careful running git clean -df. If you don't understand what it does, you might be deleting files you mean to keep, like robots.txt, uploaded files, etc. – ctlockey Jan 28 '15 at 14:57

39 As @ctlockey said, the first command also **delete directories if they are composed of ignored files only**... Lost a whole bunch of configuration files on my project :( Be careful. – Maxime Lorant Jul 16 '15 at 8:00

44

As you type git status, **(use "git checkout -- ..." to discard changes in working directory)** is shown.

e.g. `git checkout -- .`

edited Jan 7 '17 at 1:28

Dorian

**15.5k** 4 86 96

answered May 17 '16 at 11:27

Erdem ÖZDEMİR

**706** 7 5

1 Downvoted because it doesn't help to quickly discard all files. The three dots indicate that you are required to list all the files. This is especially bad if you need to discard tons of files at once, eg. during a large merge after you have staged all the modifications you like to keep – usr-local-ΕΨΗΕΛΩΝ Jun 9 '16 at 15:26

2 Of course, the correct command is "git checkout -- ." a single dot. In the comment, the three dots were a grammatical thing, to indicate there are many other options that could have been used.. – Josef.B Sep 20 '16 at 11:01

1

Just use:

```
git stash -k -u
```

This will stash **unstaged changes** and **untracked files (new files)** and keep staged files.

It's better than `reset` / `checkout` / `clean` , because you might want them back later (by `git stash pop`). Keeping them in the stash is better

answered Dec 23 '16 at 17:33

李岡諭
**2,176** 1 19 44

---

21

Just use:

```
git stash -u
```

Done. Easy.

If you *really* care about your stash stack then you can follow with `git stash drop`. But at that point you're better off using (from Mariusz Nowak):

```
git checkout -- .
git clean -df
```

Nonetheless, I like `git stash -u` the best because it "discards" all tracked and untracked changes in just *one command*. Yet `git checkout -- .` only discards tracked changes, and `git clean -df` only discards untracked changes... and typing both commands is *far* too much work :)

answered Sep 8 '16 at 6:19

Ben Wilde
**4,474** 1 27 33

---

11

In my opinion,

```
git clean -df
```

should do the trick. As per Git documentation on git clean

> git-clean - Remove untracked files from the working tree

> Cleans the working tree by recursively removing files that are not under version control, starting from the current directory.
>
> Normally, only files unknown to Git are removed, but if the -x option is specified, ignored files are also removed. This can, for example, be useful to remove all build products.
>
> If any optional ... arguments are given, only those paths are affected.

**Options**

> -d Remove untracked directories in addition to untracked files. If an untracked directory is managed by a different Git repository, it is not removed by default. Use -f option twice if you really want to remove such a directory.
>
> -f --force If the Git configuration variable clean.requireForce is not set to false, git clean will refuse to run unless given -f, -n or -i.

answered Jul 14 '16 at 7:03

Lahiru
**1,444** 4 28 28

---

35

Instead of discarding changes, I reset my remote to the origin. Note - this method is to completely restore your folder to that of the repo.

So I do this to make sure they don't sit there when I git reset (later - excludes gitignores on the Origin/branchname)

NOTE: If you want to keep files not yet tracked, but not in GITIGNORE you may wish to skip this step, as it will Wipe these untracked files not found on your remote repository (thanks @XtrmJosh).

```
git add --all
```

Then I

```
git fetch --all
```

Then I reset to origin

```
git reset --hard origin/branchname
```

Updated per user comment below: Variation to reset the to whatever current branch the user is on.

```
git reset --hard @{u}
```

edited Jan 21 '16 at 18:04

answered Aug 7 '15 at 21:15

Nick
**953** 9 19

2 A nice little variation of this I like is `git reset --hard @{u}` which resets the branch to wherever the current remote-tracking branch is – user2221343 Jan 6 '16 at 19:39

5

None of the solutions work if you just changed the *permissions* of a file (this is on DOS/Windoze)

```
Mon 23/11/2015-15:16:34.80 C:\...\work\checkout\slf4j+> git status
On branch SLF4J_1.5.3
Changes not staged for commit:
  (use "git add ..." to update what will be committed)
  (use "git checkout -- ..." to discard changes in working directory)

        modified:   .gitignore
        modified:   LICENSE.txt
        modified:   TODO.txt
        modified:   codeStyle.xml
        modified:   pom.xml
        modified:   version.pl

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

Mon 23/11/2015-15:16:37.87 C:\...\work\checkout\slf4j+> git diff
diff --git a/.gitignore b/.gitignore
old mode 100644
new mode 100755
diff --git a/LICENSE.txt b/LICENSE.txt
old mode 100644
new mode 100755
diff --git a/TODO.txt b/TODO.txt
old mode 100644
new mode 100755
```

```
diff --git a/pom.xml b/pom.xml
old mode 100644
new mode 100755
diff --git a/version.pl b/version.pl
old mode 100644
new mode 100755

Mon 23/11/2015-15:16:45.22 C:\...\work\checkout\slf4j+> git reset --hard HEAD
HEAD is now at 8fa8488 12133-CHIXMISSINGMESSAGES MALCOLMBOEKHOFF 20141223124940 Added
.gitignore

Mon 23/11/2015-15:16:47.42 C:\...\work\checkout\slf4j+> git clean -f

Mon 23/11/2015-15:16:53.49 C:\...\work\checkout\slf4j+> git stash save -u
Saved working directory and index state WIP on SLF4J_1.5.3: 8fa8488 12133-
CHIXMISSINGMESSAGES MALCOLMBOEKHOFF 20141223124940 Added .gitignore
HEAD is now at 8fa8488 12133-CHIXMISSINGMESSAGES MALCOLMBOEKHOFF 20141223124940 Added
.gitignore

Mon 23/11/2015-15:17:00.40 C:\...\work\checkout\slf4j+> git stash drop
Dropped refs/stash@{0} (cb4966e9b1e9c9d8daa79ab94edc0c1442a294dd)

Mon 23/11/2015-15:17:06.75 C:\...\work\checkout\slf4j+> git stash drop
Dropped refs/stash@{0} (e6c49c470f433ce344e305c5b778e810625d0529)

Mon 23/11/2015-15:17:08.90 C:\...\work\checkout\slf4j+> git stash drop
No stash found.

Mon 23/11/2015-15:17:15.21 C:\...\work\checkout\slf4j+> git checkout -- .

Mon 23/11/2015-15:22:00.68 C:\...\work\checkout\slf4j+> git checkout -f -- .

Mon 23/11/2015-15:22:04.53 C:\...\work\checkout\slf4j+> git status
On branch SLF4J_1.5.3
Changes not staged for commit:
  (use "git add ..." to update what will be committed)
  (use "git checkout -- ..." to discard changes in working directory)

        modified:   .gitignore
        modified:   LICENSE.txt
        modified:   TODO.txt
        modified:   codeStyle.xml
        modified:   pom.xml
        modified:   version.pl

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

```
old mode 100644
new mode 100755
diff --git a/LICENSE.txt b/LICENSE.txt
old mode 100644
new mode 100755
diff --git a/TODO.txt b/TODO.txt
old mode 100644
new mode 100755
diff --git a/codeStyle.xml b/codeStyle.xml
old mode 100644
new mode 100755
diff --git a/pom.xml b/pom.xml
old mode 100644
new mode 100755
diff --git a/version.pl b/version.pl
old mode 100644
new mode 100755
```

The only way to fix this is to manually reset the permissions on the changed files:

```
Mon 23/11/2015-15:25:43.79 C:\...\work\checkout\slf4j+> git status -s | egrep "^ M" |
cut -c4- | for /f "usebackq tokens=* delims=" %A in (`more`) do chmod 644 %~A

Mon 23/11/2015-15:25:55.37 C:\...\work\checkout\slf4j+> git status
On branch SLF4J_1.5.3
nothing to commit, working directory clean

Mon 23/11/2015-15:25:59.28 C:\...\work\checkout\slf4j+>

Mon 23/11/2015-15:26:31.12 C:\...\work\checkout\slf4j+> git diff
```

answered Nov 23 '15 at 4:30



Malcolm Boekhoff

**693** 9 6

38

You can use git stash - if something goes wrong, you can still revert from the stash. Similar to some other answer here, but this one also removes all unstaged files and also all unstaged deletes:

if you check that everything is OK, throw the stash away:

```
git stash drop
```

The answer from Bilal Maqsood with `git clean` also worked for me, but with the stash I have more control - if I do sth accidentally, I can still get my changes back

**UPDATE**

I think there is 1 more change (don't know why this worked for me before):

`git add . -A` instead of `git add .`

without the `-A` the removed files will not be staged

edited Oct 21 '15 at 8:37

answered Sep 11 '15 at 11:59

Asped

**2,793** 3 25 47

---

16

This works even in directories that are; outside of normal git permissions.

```
sudo chmod -R 664 ./* && git checkout -- . && git clean -dfx
```

Happened to me recently

edited Sep 28 '15 at 13:29

answered Sep 5 '13 at 9:38

GlassGhost

**9,954** 5 24 41

---

6

If all the staged files were actually committed, then the branch can simply be reset e.g. from your GUI with about three mouse clicks: **Branch**, **Reset**, **Yes**!

So what I often do in practice to revert unwanted local changes is to commit all the good stuff, and then reset the branch.

If the good stuff is committed in a single commit, then you can use "amend last commit" to bring it back to being staged or unstaged if you'd ultimately like to commit it a little differently.

This might not be the technical solution you are looking for to your problem, but I find it a very practical solution. It allows you to discard unstaged changes selectively, resetting the changes you don't like and keeping the ones you do.

So in summary, I simply do **commit**, **branch reset**, and **amend last commit**.

edited Sep 4 '15 at 9:56

answered Mar 20 '15 at 15:38

Ivan

**2,289** 20 20

---

21

simply say

```
git stash
```

It will remove all your local changes. You also can use later by saying

```
git stash apply
```

or git stash pop

edited Jun 30 '15 at 22:12

BeC

**180** 1 2 14

answered Apr 24 '15 at 12:19

piyushmandovra

**3,287** 2 14 26

---

239

```
git clean -df
```

Cleans the working tree by recursively removing files that are not under version control, starting from the current directory.

`-d` : Remove untracked directories in addition to untracked files

edited May 15 '15 at 13:42

falsarella
**10.7k** 7 56 97

answered Dec 7 '11 at 13:09

Elvis Ciotti
**3,940** 1 18 17

---

7

When you want to transfer a stash to someone else:

```
# add files
git add .
# diff all the changes to a file
git diff --staged > ~/mijn-fix.diff
# remove local changes
git reset && git checkout .
# (later you can re-apply the diff:)
git apply ~/mijn-fix.diff
```

[edit] as commented, it ís possible to name stashes. Well, use this if you want to share your stash ;)

edited Apr 20 '15 at 10:39

answered Jul 8 '13 at 15:07

twicejr
**1,133** 1 11 18

5 Actually Git stash can have a title. For instance `git stash save "Feature X work in progress" .` – Colin D Bennett Dec 9 '14 at 22:39

---

103

My favorite is

```
git checkout -p
```

That lets you selectively revert chunks.

See also:

```
git add -p
```

Ben
**4,770** 1 24 29

9 I love the ability to see the actual change before it's discarded. – Penghe Geng Feb 3 '15 at 21:42

2 I've never thought about. That `-p` adds a nice extra layer of safety. Combine it with `git clean -d` to actually answer OP. – Stephan Henningsen Apr 27 '16 at 6:39

14

```
cd path_to_project_folder  # take you to your project folder/working directory
git checkout .             # removes all unstaged changes in working directory
```

edited May 31 '14 at 10:04

answered May 30 '14 at 9:26

vivekporwal04
**405** 3 14

43

```
git checkout -f
```

```
man git-checkout :

-f, --force
```

When switching branches, proceed even if the index or the working tree differs from HEAD. This is used to throw away local changes.

When checking out paths from the index, do not fail upon unmerged entries; instead, unmerged entries are ignored.

answered May 17 '14 at 2:28

Bijan
**5,549** 5 28 25

2 This would discard changes in the index!! (And the OP requires to leave them as is.) – Robert Siemer Apr 19 '15 at 13:59

9

another repo. Short answer: delete fork and refork, but **read the warnings on github**.

I had a similar problem, perhaps not identical, and I'm sad to say my solution is not ideal, but it is ultimately effective.

I would often have git status messages like this (involving at least 2/4 files):

```
$ git status
# Not currently on any branch.
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   doc/PROJECT/MEDIUM/ATS-constraint/constraint_s2var.dats
#       modified:   doc/PROJECT/MEDIUM/ATS-constraint/parsing/parsing_s2var.dats
#
# 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:   doc/PROJECT/MEDIUM/ATS-constraint/constraint_s2Var.dats
#       modified:   doc/PROJECT/MEDIUM/ATS-constraint/parsing/parsing_s2Var.dats
```

A keen eye will note that these files have dopplegangers that are a single letter in case off. Somehow, and I have no idea what led me down this path to start with (as I was not working with these files myself from the upstream repo), I had switched these files. Try the many solutions listed on this page (and other pages) did not seem to help.

**I was able to fix the problem by deleting my forked repository and all local repositories, and reforking. This alone was not enough; upstream had to rename the files in question to new filenames.** As long as you don't have any uncommited work, no wikis, and no issues that diverge from the upstream repository, you should be just fine. Upstream may not be very happy with you, to say the least. As for my problem, it is undoubtedly a user error as I'm not that proficient with git, but the fact that it is far from easy to fix points to an issue with git as well.

edited Jan 13 '14 at 16:42

answered Jan 5 '14 at 4:53

bbarker

**4,713** 3 23 36

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?