Haramohan Sahu

PERFORCE BASIC COMMANDS WITH EXAMPLES

Haramohan Sahu

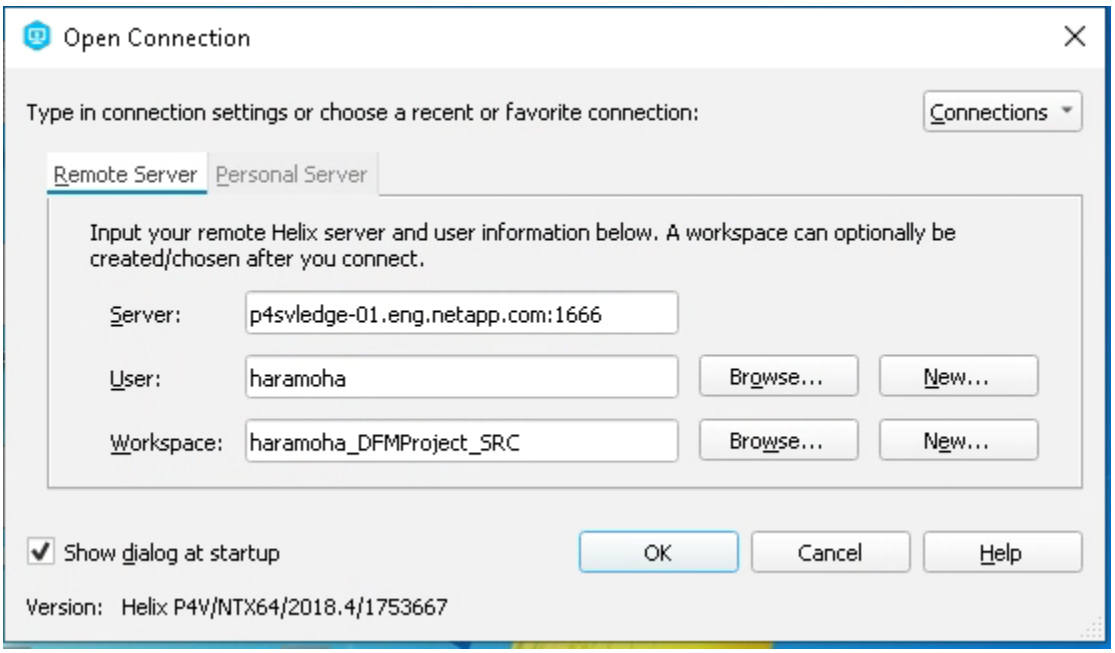
CALSOFT Bangalore

Contents

Perforce client

A **Perforce client** workspace is a set of files on a user's machine that mirror a subset of the files in the depot. More precisely, it is a named mapping of depot files to workspace files. Use the p4 **client** command to create or edit a **client** workspace specification.

Perforce gui setup

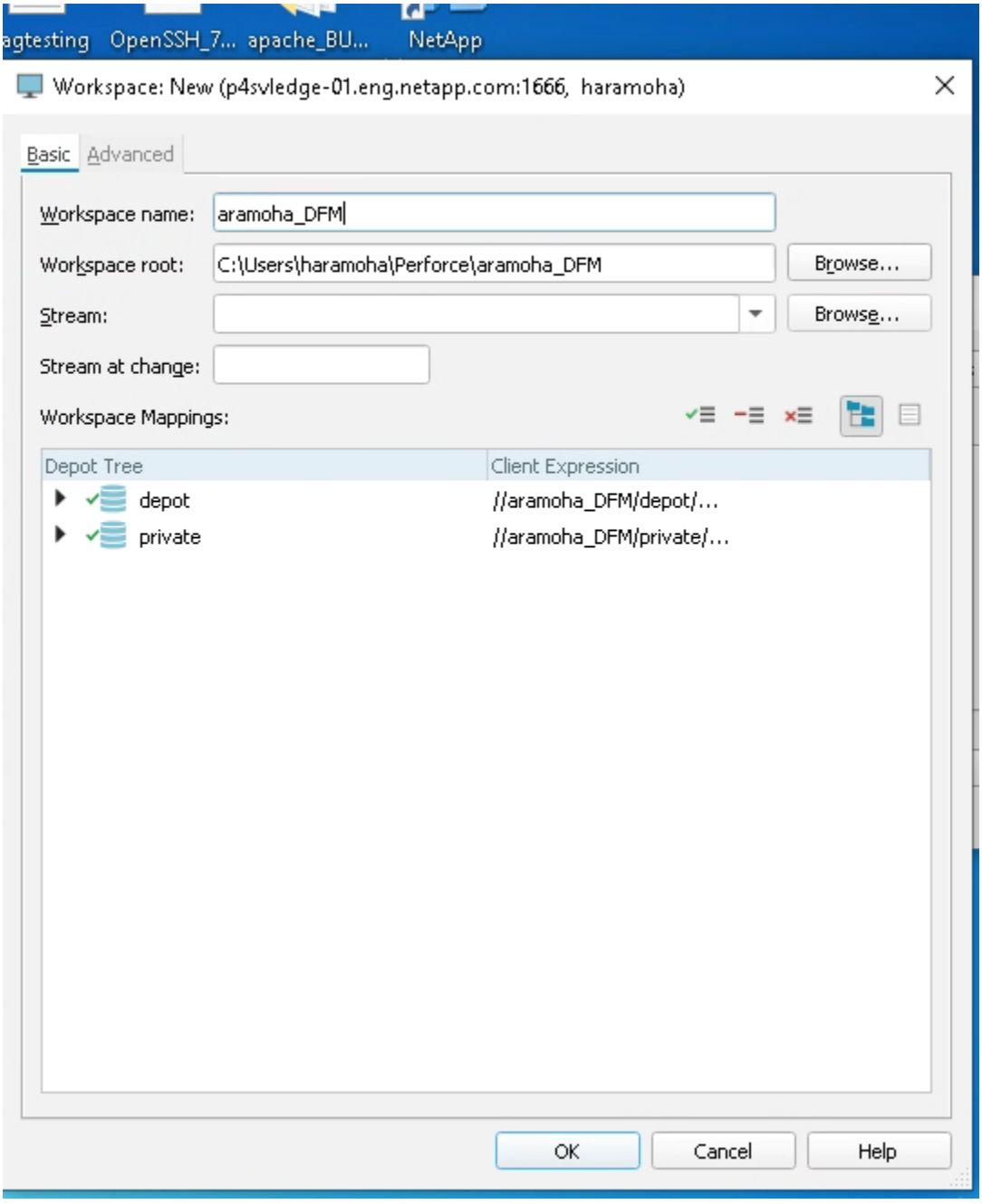


Server: hostname:port → perforce port will always 1666

User: It will be created by Perforce admin

Workspace: click on new

How to create new workspace in GUI

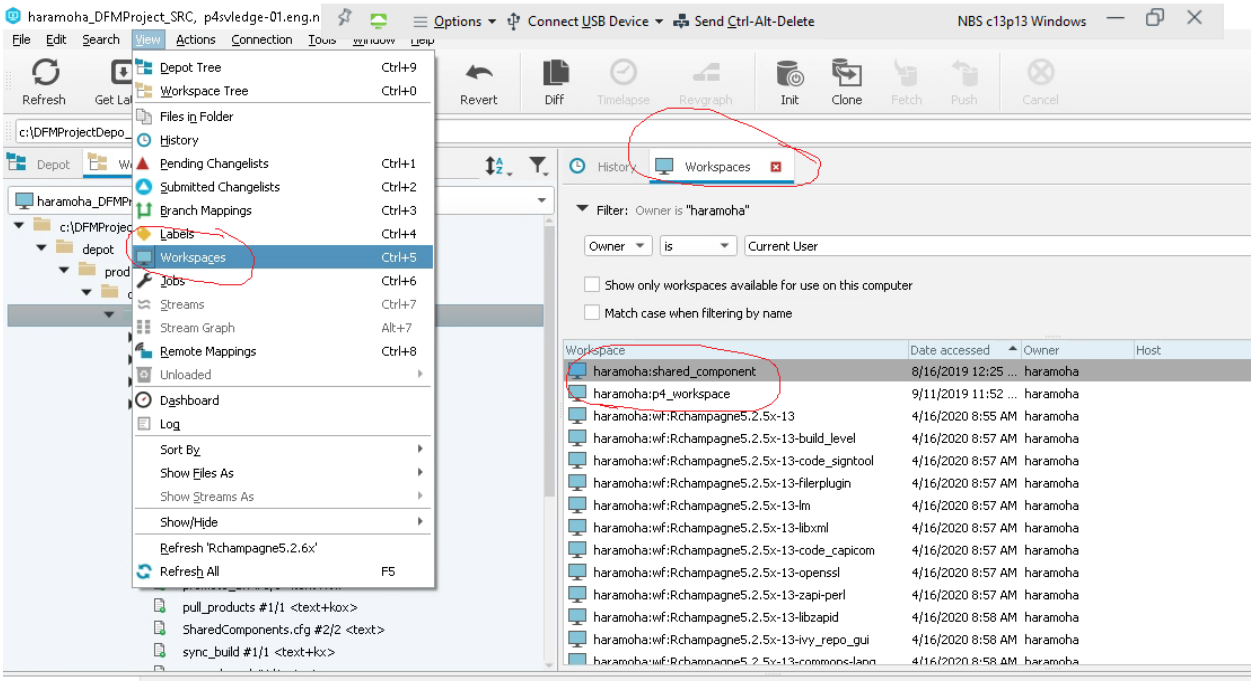


Workspace name = always write username_ProjectName

Workspace root: create folder in “C:\drive” → where depot files will be stored

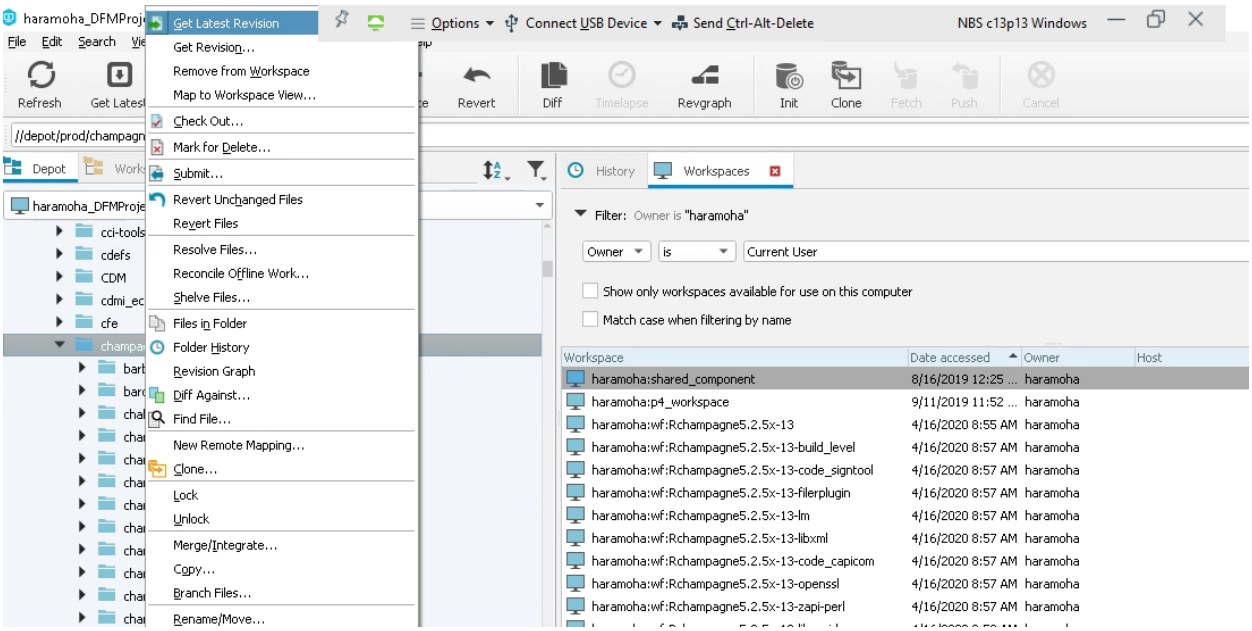
Then click ok → your workspace created.

How to select Workspace in GUI



You want to see your workspace, then click on view then select workspace, It will show your workspace

How to get files to your workspace in GUI

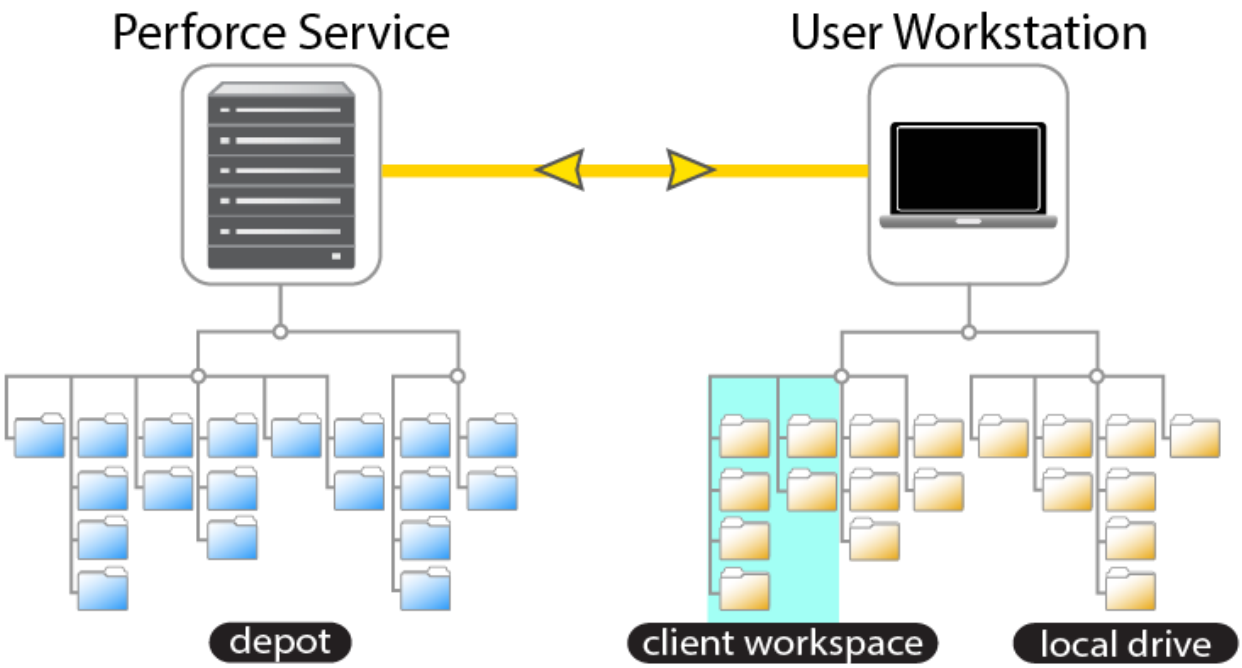


Select the Depot tab, then select the component the right click the get latest revision, it will copy files to your workspace

Perforce applications

You use Perforce applications to communicate with the versioning service. Perforce applications enable you to check files in and out, manage conflicts, create development branches, track bugs and change requests, and more. Perforce applications include:

- `P4`, the Perforce Command-Line Client, for all platforms
- `P4V`, the Perforce Visual Client, for Mac OS X, UNIX, Linux, and Windows
- `P4Web`, the Perforce Web Client, a browser-based interface to Perforce



Mapping files in the depot to your workspace

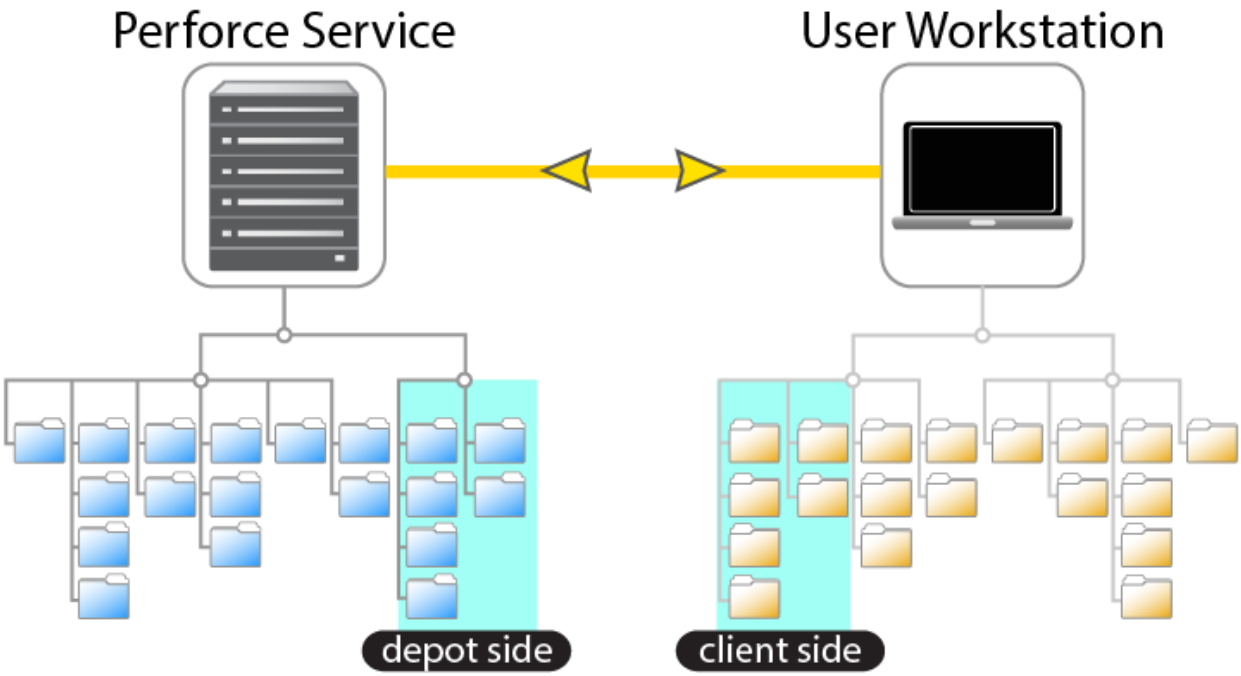
- Workspace defines a mapping of files between server and the client. This is nothing but which files you want to download.
- Workspace definition are stored on the perforce server, not on the developer's machine.
- Workspace can be updated, modified after it is created.

To control where the depot files appear under your workspace root, you must map the files and directories on the shared versioning service to corresponding areas of your local hard drive. These mappings constitute your *workspace view*.

Workspace views:

- Determine which files in the depot can appear in a workspace.
- Map files in the depot to files in the workspace.

Client workspace views consist of one or more lines, or *mappings*. Each line in your workspace view has two sides: a *depot side* that designates a subset of files within the depot and a *client side* that controls where the files specified on the depot side are located under your workspace root.



Creating a workspace doesn't transfer any files from the depot to your computer. The workspace and its view merely define the mapping that controls the relationship between the depot and your workspace when files are transferred.

Getting files from the DEPO

The perforce manages the depot, which is nothing but a shared repository, that holds the all the versions of every files.

Your workspace is nothing but the subset of depot's files, this is mapped to depots files.

`p4 client` command to create or edit a client workspace specification.

For new workspaces, the client name defaults to the **P4CLIENT** environment variable if set, or to the current host name.

The command `p4 workspace` is an alias for `p4 client`.

P4 client `-d clientname`

Example:

```
$ P4 client –d haramoha:shared_component
```

Basic Environment Variables

p4 set **P4CLIENT**= value

`P4CLIENT` => Name of current client workspace.

`P4PORT` → `p4 -p protocol:host:port`

`P4PASSWD`→ `p4 -P passwd`

`P4USER`→ `p4 -u username`

cat P4ENV

P4CLIENT=haramoha:wf:Rchampagne5.2.6x-02

P4PORT=p4-1666.eng.netapp.com:1666

cat p4Client.tmp

Client: haramoha:wf:Rchampagne5.2.6x-02

Owner: haramoha

Description:

Rchampagne5.2.6x WinFarm Client

Root: /x/eng/build9/scratch/haramoha/p4/aw/Rchampagne5.2.6x-02

Options: noallwrite noclobber nocompress crlf unlocked nomodtime normdir

View:

//depot/prod/champagne/Rchampagne5.2.6x/... //haramoha:wf:Rchampagne5.2.6x-02/...

cat p4Client.out

Client haramoha:wf:Rchampagne5.2.6x-02 saved.

cat P4CLIENTS

haramoha:wf:Rchampagne5.2.6x-02

haramoha:wf:Rchampagne5.2.6x-02-build_level

haramoha:wf:Rchampagne5.2.6x-02-code_signtool

haramoha:wf:Rchampagne5.2.6x-02-filerplugin

haramoha:wf:Rchampagne5.2.6x-02-lm

haramoha:wf:Rchampagne5.2.6x-02-libxml

haramoha:wf:Rchampagne5.2.6x-02-code_capicom

haramoha:wf:Rchampagne5.2.6x-02-zapi-perl

haramoha:wf:Rchampagne5.2.6x-02-openssl

haramoha:wf:Rchampagne5.2.6x-02-libzapid

haramoha:wf:Rchampagne5.2.6x-02-ivy_repo_gui

haramoha:wf:Rchampagne5.2.6x-02-commons-lang

haramoha:wf:Rchampagne5.2.6x-02-filer_adminapi

haramoha:wf:Rchampagne5.2.6x-02-libadt

haramoha:wf:Rchampagne5.2.6x-02-xml-builder

haramoha:wf:Rchampagne5.2.6x-02-wsdl

haramoha:wf:Rchampagne5.2.6x-02-apache

haramoha:wf:Rchampagne5.2.6x-02-log4j

haramoha:wf:Rchampagne5.2.6x-02-ontaperrs

haramoha:wf:Rchampagne5.2.6x-02-apitest

haramoha:wf:Rchampagne5.2.6x-02-zephyr

haramoha:wf:Rchampagne5.2.6x-02-handlergen

haramoha:wf:Rchampagne5.2.6x-02-hs-wsdl

haramoha:wf:Rchampagne5.2.6x-02-dfm-zapidoc

haramoha:wf:Rchampagne5.2.6x-02-jetty

haramoha:wf:Rchampagne5.2.6x-02-libnetapp

haramoha:wf:Rchampagne5.2.6x-02-openssh

haramoha:wf:Rchampagne5.2.6x-02-ihub3.1.1

haramoha:wf:Rchampagne5.2.6x-02-code_sign_cert

haramoha:wf:Rchampagne5.2.6x-02-putty

haramoha:wf:Rchampagne5.2.6x-02-ntapadmin

haramoha:wf:Rchampagne5.2.6x-02-champagne_shared

[haramoha@cycrh6svl21 ~/aw_Rchampagne5.2.6x-02]$

P4client

The `p4 client` form contains a number of fields; at this point, the important fields are the `Root:` and `View:` fields:

Field	Meaning
`Root:`	Your *client workspace root* is the topmost subdirectory of your client workspace. Set your client workspace root to a directory under your control; you will be doing most of your development work on files located in your client workspace.
`View:`	The *client workspace view* determines which files and directories in the depot are mapped to your client workspace, and where the files appear beneath your client workspace root.

Note	In the text forms used by Perforce, field names always start in the leftmost column of text, and field values are indented with tabs or spaces. Perforce requires that there be at least one space or a tab prior to the contents of a form field.

Example: *Setting the client workspace root and the client workspace view: Ed is working with his Elm files in a setting as described above. He's set the environment variable* `P4CLIENT` *to* `eds_elm`*. He then types* `p4 client` *from his home directory, and sees the following form:*

```
Client: eds_elm
Owner:  edk
Description:
        Created by edk.
Root:   /usr/edk
Options:        nomodtime noclobber
View:
        //depot/...    //eds_elm/...
```

Copying depot files into your workspace

By default, `p4 sync` updates your entire client workspace. To limit the update to only a portion of your client workspace, you can supply filenames or wildcards to `p4 sync`.

> **Example:** *Copying files from the depot to a client workspace.*
> *Lisa is responsible for the documentation for the Elm project. Her client workspace root is a directory called `elm_ws`, and she has set her client workspace view to include only the `elm_proj` documentation tree. To populate her client workspace, she enters her client workspace root, runs `p4 sync`, and sees:*

```
$ cd ~/elm_ws
$ p4 sync
//depot/elm_proj/doc/elmdoc.0#2 - added as
/usr/lisag/elm_ws/doc/elmdoc.0
//depot/elm_proj/doc/elmdoc.1#2 - added as
/usr/lisag/elm_ws/doc/elmdoc.1
<etc.>
```

> *After the `p4 sync` command completes, the most recent revisions of the Elm project's documentation files as mapped through Lisa's client workspace view, are available in her workspace.*

Adding files to the depot

To add a file (or files) to the depot, type `p4 add filename(s)`. The `p4 add` command opens the file(s) for `add` and includes the files in the default changelist.

After you have added files to the changelist, you must submit your changelist to the depot by using the `p4 submit` command.

> **Example:** *Adding files to a changelist.*
> *Ed is writing a help manual for Elm. The files are named `elmdoc.0` through `elmdoc.3`, and they're sitting in the `doc` subdirectory of his client workspace root. He wants to add these files to the depot.*

```
$ cd ~/elm/doc
$ p4 add elmdoc.*
//depot/elm_proj/doc/elmdoc.0#1 - opened for add
//depot/elm_proj/doc/elmdoc.1#1 - opened for add
//depot/elm_proj/doc/elmdoc.2#1 - opened for add
//depot/elm_proj/doc/elmdoc.3#1 - opened for add
```

Adding more than one group of files at once

You can supply multiple file arguments on the command line. For example:

Example: *Using multiple file arguments on a single command line.*
Ed wants to add all of his Elm library, documentation, and header files to the depot.

```
$ cd ~
$ p4 add elm/lib/* elm/hdrs/* elm/doc/*
//depot/elm_proj/lib/Makefile.SH#1 - opened for add
//depot/elm_proj/lib/add_site.c#1 - opened for add
//depot/elm_proj/lib/addrmchusr.c#1 - opened for add
<etc.>
```

Editing files in the depot

Example: *Opening a file for `edit`:*
Ed wants to make changes to his `elmdoc.3` file. He opens the file for `edit`.

```
$ cd ~/elm
$ p4 edit doc/elmdoc.3
//depot/elm_proj/doc/elmdoc.3#1 - opened for edit
```

Deleting files from the depot

To delete a file from the depot, use the `p4 delete` command to open the file for `delete` in the default changelist.

The `p4 delete` command deletes the file from your client workspace as soon as you run the `p4 delete` command, but deletion of the file in the depot does not occur until you use `p4 submit` to submit the changelist containing the delete operation to the depot.

Example: *Deleting a file from the depot.*
Ed's file `doc/elmdoc.3` is no longer needed. He deletes it from both his client workspace and from the depot as follows:

```
$ cd ~/elm/doc
$ p4 delete elmdoc.3
//depot/elm_proj/doc/elmdoc.3#1 - opened for delete
```

Submitting your changes to the depot

Submitting a changelist to the depot works atomically: either all the files in the changelist are updated in the depot, or none of them are. (In Perforce terminology, this is called an *atomic change transaction*).

To submit a changelist, use the `p4 submit` command.

Example: *Adding, updating, and deleting files in a single changelist:*
Ed is writing the portion of Elm that is responsible for multiple folders (multiple mailboxes). He has added a new source file `src/newmbox.c`*, and he needs to edit the header file* `hdrs/s_elm.h` *and some of the* `doc/elmdoc` *files to reflect his changes.*
Ed adds the new file and prepares to edit the existing files as follows:

```
$ cd ~
$ p4 add elm/src/newmbox.c
//depot/elm_proj/src/newmbox.c#1 - opened for add
<etc.>
$ p4 edit elm/hdrs/s_elm.h doc/elmdoc.*
//depot/elm_proj/hdrs/s_elm.h#1 - opened for edit
//depot/elm_proj/doc/elmdoc.0#1 - opened for edit
//depot/elm_proj/doc/elmdoc.1#1 - opened for edit
//depot/elm_proj/doc/elmdoc.2#2 - opened for edit
```

He edits the header file and the documentation files in his workspace. When he is ready to submit the changelist, he types `p4 submit` *and sees the following form in a standard text editor:*

```
Change: new
Client: eds_elm
User:   edk
Status: new
Description:
        <enter description here>
Files:
        //depot/elm_proj/doc/elmdoc.0  # edit
        //depot/elm_proj/doc/elmdoc.1  # edit
        //depot/elm_proj/doc/elmdoc.2  # edit
        //depot/elm_proj/hdrs/s_elm.h    # edit
        //depot/elm_proj/src/newmbox.c   # add
```

Ed changes the contents of the `Description:` *field in the* `p4 submit` *form to describe the changes he's made as follows:*

```
Change: new
Client: eds_elm
User:   edk
Status: new
Description:
        Changes to Elm's mailbox functionality
Files:
        //depot/elm_proj/doc/elmdoc.0  # edit
        //depot/elm_proj/doc/elmdoc.1  # edit
        //depot/elm_proj/doc/elmdoc.2  # edit
        //depot/elm_proj/hdrs/s_elm.h    # edit
        //depot/elm_proj/src/newmbox.c   # add
```

All of Ed's changes are grouped together in a single changelist. When Ed quits from the editor, either all of these files are updated in the depot, or, if the submission fails for any reason, none of them are.

The `p4 submit` form includes a `Description:` field; you must supply a description in order for your changelist to be accepted.

The `Files:` field contains the list of files in the changelist. If you remove files from this field, any files not included in the changelist are carried over to a new default changelist, and reappear the next time you use `p4 submit`.

If a directory containing a new file does not exist in the depot, the directory is automatically created in the depot when you submit the changelist.

When you run `p4 submit`, the operating system's write permission on the submitted files in your client workspace is turned off. Write permissions are restored when you open a file for editing with `p4 edit`.

Backing out: reverting files to their unopened states

Example: *Reverting a file back to the last synced version.*
Ed wants to edit a set of files in his src directory: `leavembox.c`, `limit.c`, *and* `signals.c`. *He opens the files for* `edit`*:*

```
$ cd ~elm/src
$ p4 edit leavembox.c limit.c signals.c
//depot/elm_proj/src/leavembox.c#2 - opened for edit
//depot/elm_proj/src/limit.c#2 - opened for edit
//depot/elm_proj/src/signals.c#1 - opened for edit
```

and then realizes that `signals.c` *is not one of the files he plans to change, and that he did not intend to open it. He reverts* `signals.c` *to its unopened state with* `p4 revert`*:*

```
$ p4 revert signals.c
//depot/elm_proj/src/signals.c#1 - was edit, reverted
```

If you use `p4 revert` on a file you opened with `p4 delete`, the file reappears in your client workspace immediately. If you revert a file opened with `p4 add`, the file is removed from the changelist, but is left intact in your client workspace.

If you revert a file you originally opened with `p4 edit`, the last synced version is written back to the client workspace, overwriting the edited version of the file. To reduce the risk of accidentally overwriting your changes, you can preview any revert by using `p4 revert -n` before running `p4 revert`. The `-n` option reports what files would be reverted by `p4 revert` without actually reverting the files.

Perforce	Description
client/workspace	read-only files, all meta info is in server depot
depot	central repository
Change list	changes to a set of files, depot-global
base version	the version a file is based on
head	the latest change in a branch, new changes must be based on this
branch	location in depot dedicated to one project
Working on files	changes are pending in WD until commit
p4 edit	open a file for edit
p4 add	add files
p4 delete	remove files
p4 diff	show diff for files opened for edit
$P4DIFF	external (graphical) diff program
p4 opened	show scheduled changes
p4 integ; p4 delete	move file keeping history
p4 integ	copy file keeping history
p4 revert	drop all changes, revert to base version
p4 sync	update p4 client to other repo version
Looking in repository	changes nothing
p4 print	show repository file
p4 annotate	show modification info for each line in a file
p4 changes	show history of (branch of) repo
p4 filelog	show history of file
p4 tag	add a tag for a changeset
p4 have	show info for client files

p4 have	show info for client files
p4 have	show info for client files
p4 files	show info for repo files
p4 branches	list branch specifications - that is officially related branches
p4 describe	show changelist with its changes
Working with repository	all changes are pending until submit
p4 client	create a new branch+client from scratch
p4 client; p4 sync	create a new workspace for existing repo
p4 client; p4 integ	create a new branch+client based on other
p4 integ; p4 resolve	get changes from other p4 branch
p4 integ	integrate changes from other branch
p4 integ	integrate changes to other branch
(diff; merge)	can't move changes to other repos - must be done manually
p4 sync; p4 resolve	rebase scheduled changes to other repo version
p4 sync	get new changes from central repository
p4 sync; p4 resolve	get changes from central repo and merge them with local changes
p4 submit	add pending changes to the chain of changesets in branch
p4 submit	submit a change to the central repository
p4 triggers	run commands (send email) when certain events happen
p4 protect	control user's access

To Understand in detail:

https://www.perforce.com/perforce/doc.051/manuals/p4guide/index.html