# Proposal to Use Version Control to Manage Consulting Artifacts

Dave Nicolette

10 April 2017

Table of Contents

[Proposal to Use Version Control to Manage Consulting Artifacts 1](#_Toc479588994)

[Introduction 1](#_Toc479588995)

[Problems 1](#_Toc479588996)

[Problem 1: Difficulty locating the correct version of an artifact 1](#_Toc479588997)

[Problem 2: References to other clients are not always removed 2](#_Toc479588998)

[Problem 3: Changes are lost or overwritten when more than one person makes updates 2](#_Toc479588999)

[Problem 4: Concierge approach substitutes new problems for the old 2](#_Toc479589000)

[General Solution 2](#_Toc479589001)

[Challenges 3](#_Toc479589002)

[Challenge 1: Version control systems offer too many powerful options 3](#_Toc479589003)

[Challenge 2: Version control systems do not protect users against themselves 4](#_Toc479589004)

[Overcoming the challenges 4](#_Toc479589005)

[Proposed solution 4](#_Toc479589006)

[Solution: Organization 4](#_Toc479589007)

[Solution: Workflow 5](#_Toc479589008)

[Solution: Safety 7](#_Toc479589009)

[Summary 7](#_Toc479589010)

[Sample script for Unix/Linux/Mac OSX 8](#_Toc479589011)

[Setup for Mac OSX 10](#_Toc479589012)

## Introduction

This document describes an approach for maintaining consulting documents under version control. The proposed approach is based on the version control system, Git.

## Problems

LeadingAgile has a set of documentation artifacts that support our governance model and organizational transformation approach. These include templates for backlog management such as the Epic Canvas, User Story, and others, as well as guides for using the artifacts. Also included are presentation decks for our transformation workshop and various training sessions. The documentation could be extended to include boilerplate SOWs and other items that are more-or-less common across engagements.

### Problem 1: Difficulty locating the correct version of an artifact

There is no easy-to-find, easy-to-use source for the “correct” or “Gold Standard” versions of these artifacts. Multiple versions of documents have proliferated. Some consultants maintain their own personal copies which diverge over time from the company standard. Other consultants know nothing about these personal versions and cannot make use of them.

We have attempted various methods to control the proliferation of documents and to provide the correct versions to consultants. To date, these have not simplified the process or provided any real assurance to consultants that they have the correct version of a document in hand. Consultants try to locate a version of each artifact that seems reasonable to them, but they are not sure whether they have the right version.

### Problem 2: References to other clients are not always removed

Using Dropbox, SharePoint, Google Docs, or similar tools on a self-service basis has resulted in numerous different versions of the same artifact. Many of these contain references to clients, and these references are not always removed when a document is re-used for another engagement. This is a result of copying and modifying artifacts from previous engagements rather than starting with a clean base version for each engagement.

### Problem 3: Changes are lost or overwritten when more than one person makes updates

When consultants are working collaboratively on changes to the documents, there is no convenient way to keep all the modifications straight. Typically, they create multiple copies of the documents with some sort of naming convention that suggests modification dates or informal version numbers or that include the initials of the person who made the modifications. It is easy to lose track of some of the modifications, or to end up with overlapping or contradictory content. This contributes to the proliferation of copies of the same document.

### Problem 4: Concierge approach substitutes new problems for the old

An alternative approach has been to establish a “concierge service” whereby the Gold Standard documents are controlled by a handful of authorized people. Consultants request the Gold Standard artifacts they need, and these are provided under the terms of a service level agreement. This approach introduces administrative overhead in the hope of avoiding errors, reflecting traditional (that is, non-agile) thinking. In addition, the full set of Gold Standard artifacts has not yet been finalized, and some documents are not available through this mechanism.

Another issue is that consultants usually want an artifact immediately, and do not wish to wait for turnaround by a separate team. They deal with the situation by using whatever version they can find quickly in Dropbox or other sources.

## General Solution

A *bona fide* version control system solves these problems. It is the correct type of tool for the situation.

All version control systems have some mechanism for labeling versions of files so that people who need a specific version can retrieve it without any concern for modifications that others may be making to the files, and that are not yet “in production.” They can also retrieve a previous version of the files, if necessary.

While version control systems were invented to manage software source files, the tools do not know anything about the files they are managing. Github, for example, is widely used to manage files other than software source files, including book manuscripts and sheet music. There is no reason it could not be used to manage our Gold Standard artifacts and make them easy for all consultants to locate and use.

## Challenges

It is one thing to identify version control systems as the right type of tool for the situation, and another to make it practical and easy for the intended user base to use such a tool.

The main challenge is that version control systems are meant for “technical” people. Many LeadingAgile consultants are comfortable using “office”-style tools, such as Microsoft Office, LibreOffice, or Google Docs, and uncomfortable using tools that are designed for technical people.

### Challenge 1: Version control systems offer too many powerful options

Technical tools generally make two assumptions about users that are different from general software products:

* the users need the power to change “anything” without restriction; and
* they have the knowledge to use the tool safely.

The first assumption means the tools support a long list of commands and command options that enable users to manipulate the repository of artifacts in every imaginable way.

LeadingAgile consultants only require a small subset of these commands. They may feel intimidated by the apparent complexity of a tool that presents them with so many options, most of which they do not need, and many of which are “dangerous” in the hands of naïve users.

For example, consider the following series of commands:

git clone https://github.com/leadingagile/gold-standard

pushd gold-standard

git checkout tags/gold-1.0 -b momandpopstores

curl -u "leadingagile" https://api.github.com/user/repos -d '{"name":"momandpopstores"}'

git push --mirror https://github.com/leadingagile/momandpopstores.git

popd

rm -rf gold-standard.git

git clone https://github.com/leadingagile/momandpopstores

pushd momandpopstores

echo '# momandpopstores based on gold-standard/gold-1.0' > README.md

git add README.md

git commit -m "momandpopstores based on gold-standard/gold-1.0"

git push -u origin master

popd

These commands accomplish the workflow (described below) for getting a copy of the Gold Standard artifacts and loading them into a client-specific Git repository for a client named “Mom and Pop Stores.”

To a non-technical user, this sort of thing can be intimidating. The details lie outside a non-technical consultant’s everyday working focus or central professional interest. The consultant will only need to do this occasionally; only when setting up an artifact repository for a new engagement. It is easy to forget details about commands one does not use frequently.

Any mistake could result in data loss or data corruption, or could move artifacts into a branch where the consultant cannot find them again. This level of detail must be transparent to the consultants, or they will continue to avoid using this type of tool.

### Challenge 2: Version control systems do not protect users against themselves

The tools typically do not prompt the user with helpful warnings like “This will delete all your work. Are you sure?” They just go ahead and delete all your work. Unlike tools intended for general users, tools intended for technical people assume the users will understand what the tool is doing under the covers, and that they know enough to select commands and options wisely. In addition, tools of this kind must be *scriptable*, which means they cannot pause to display interactive messages such as “Are you sure?”

It is easy to “break” things in a version control system if one saves data or changes something without understanding what the system is doing. Between the mysteries of the command line and the risk of data loss, non-technical consultants are reluctant to make use of version control systems to manage the artifacts.

## Overcoming the challenges

Consultants worry that problems would occur frequently or possibly in a non-recoverable way. They tend to remain in their comfort zone by inventing “creative” ways to use office tools to simulate version control. This leads to the problems noted above.

Office tools are not suited to the purpose. It would be preferable to overcome the challenges in using a version control system by making it easier for non-technical people to use version control effectively, easily, and safely. We can accomplish this through a combination of

* well-thought-out naming conventions; and
* scripts that wrap the raw version control commands so that users will not use a “dangerous” command or forget to include a critical command-line flag with an otherwise-safe command.

## Proposed solution

The proposed solution comprises:

* a scheme for organizing the files under version control
* a documented workflow for checking out Gold Standard artifacts and saving client-specific versions
* wrapper scripts that protect the user from accidentally overwriting data incorrectly

### Solution: Organization

Artifacts are stored in multiple Git *repositories*, or *repos*. The primary repo is the one that contains the Gold Standard artifacts. This repo is updateable by authorized individuals, and readable by everyone.

A separate repo is created for each client engagement. Consultants working on a new engagement retrieve the current Gold Standard artifacts from the main repo and create a new repo for the client. They modify the artifacts in the client repo from that point forward.

With the files organized in this way, everyone knows where to find the current Gold Standard artifacts as well as any previous versions that may still be useful.

Git supports *tags* to identify releases. The current Gold Standard set of artifacts is tagged with a known (documented) name. Consultants who need to obtain a copy to start up a new engagement can check out this version from Git using its name. If they need a previous Gold Standard version, they check it out using its tag name, as well. If they need to start fresh with one of the individual artifacts, they can check it out by itself without overwriting their copy of all the other files, or they can revert the file to a previous version.

If LeadingAgile leaders are modifying the artifacts but the changes are not ready to be used, then these versions of the files are not included in any tagged (usable) version. Consultants cannot accidentally pick up an incomplete or not-yet-blessed artifact. They would have to go out of their way to select an unreleased version deliberately, which they can do if necessary.

Consultants can use the tagging feature within any client repo to keep track of milestone versions of the artifacts. This is preferable to creating numerous, randomly-named folders under Dropbox to keep archived files. This feature gives us a simple way to store the complete set of artifacts pertaining to a SOW without losing any history or data, and without needing multiple copies of everything in Dropbox, SharePoint, or elsewhere.

### Solution: Workflow

Four version control workflows are pertinent to the needs of LeadingAgile consultants and leadership. They will rarely, if ever, need to use any additional features of the tool.

#### Workflow 1: Consultant setting up artifacts for a new engagement

1. Check out the Gold Standard artifacts from the main Git repo.
2. Create a Git repo to contain the client’s version of the artifacts.
3. Commit the Gold Standard artifacts to the client repo. (This is separate from the main repo, so no harm will befall the actual Gold Standard artifacts.)

#### Workflow 2: Consultant working on an engagement

1. Check out the client version of the artifacts from the client repo.
2. Make whatever modifications are necessary. Git allows multiple people to work on the same files at the same time without causing any data corruption. Therefore, there is no confusion and no worries about working on the wrong version of an artifact.
3. Commit the changes to the client repo.
4. Repeat as necessary.

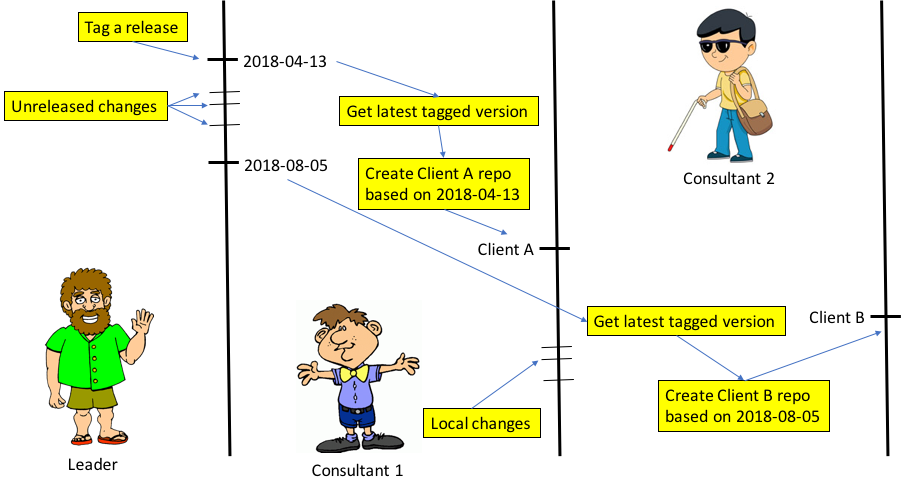
#### Workflow 3: Consultant finalizing the work for a SOW and setting up work for the next SOW, same client

1. Commit the documents to the client repo and give them a tag name that represents the completed SOW.
2. Continue working with the documents as before, using the same client repo.

#### Workflow 4: LeadingAgile leadership working on the Gold Standard artifacts

1. Ensure the latest usable version of the Gold Standard artifacts is tagged so that consultants can identify them and check them out.
2. Make whatever modifications are desired.
3. When you are satisfied with the modifications, tag the current version as the next “gold” version to make it usable to the consultants.

Leader releases a set of artifacts by tagging a commit. He continues to make changes, which are not part of the tagged release. Consultant 1 checks out the currently-tagged release, which is 2018-04-13, and uses it as the basis for a new repo for Client A. Leader releases a new version, 2018-08-05. Consultant B checks out the currently-tagged release and uses it to create a repo for Client B, based on 2018-08-05. Unreleased changes to Gold Standard artifacts and local changes in Client repos do not affect anyone else.



### Solution: Safety

To make it safe and easy for non-technical consultants to take advantage of a version control system, we need to hide the details from them and provide them with a simple way to run the handful of commands they need. It is a trivial exercise to write a script to run the commands under the covers. To illustrate this, a sample script for Workflow 1 is provided below.

With a modest additional effort, we could create a GUI front-end tailored to the needs of our consultants, rather than a general-purpose one intended for technical people.

## Summary

To use Git to manage the versions of our consulting artifacts is simple, practical, and almost cost-free. LeadingAgile already has a paid Github account that supports private repos. We can create additional repos at will with no additional cost.

We have the technical skills in house to create any necessary wrapper scripts or graphical user interfaces to make it simple and safe for non-technical consultants to use Git.

## Sample script for Unix/Linux/Mac OSX

To illustrate the practicality of scripting a few basic interactions with Github, here is a sample script for Unix/Linux/OSX systems that creates a client-specific repo and loads it with a tagged version of the Gold Standard artifacts. Scripts for the other workflows can be written, too. Similar scripts for Windows could be written, as well.

#!/bin/bash

#---------------------------------------------------------------------------------

# Sample bash script to copy the Gold Standard repo to a client-specific repo.

# The script also creates the client-specific repo.

#

# For usage help, run

# copy-latest-gold --help

#---------------------------------------------------------------------------------

# GOLD\_USER is the Github username of the owner of the Gold Standard repo

GOLD\_USER=leadingagile

# GOLD\_REPO is the Github name of the Gold Standard repo

GOLD\_REPO=gold-standard

# Don't change anything below this.

REPO=

function show\_help {

echo 'Usage: copy-latest-gold --gold gold-standard-version --client target-repository-name'

echo ' options:'

echo ' -g | --gold : Tag name for Gold Standard version to pull'

echo ' -c | --client : Git repo name to populate for the client'

echo ' -v | --verbose : Display progress'

echo " -n | --dry-run : Display steps but don't update anything (implies --verbose)"

echo " -h | --help : Display usage help (the output you're reading now)"

exit 0

}

OPTS=`getopt -o vhng:c: --long verbose,dry-run,gold:,client:,help -n 'parse-options' -- "$@"`

if [ $? != 0 ] ; then echo "Failed parsing options." >&2 ; exit 1 ; fi

echo "$OPTS"

eval set -- "$OPTS"

VERBOSE=false

DRY\_RUN=false

STACK\_SIZE=0

while true; do

case "$1" in

-g | --gold ) GOLD\_VERSION="$2"; shift; shift ;;

-c | --client ) REPO="$2"; shift; shift ;;

-v | --verbose ) VERBOSE=true; shift ;;

-h | --help ) show\_help ;;

-n | --dry-run ) DRY\_RUN=true; VERBOSE=true; shift ;;

-- ) shift; break ;;

\* ) break ;;

esac

done

[ -z "$GOLD\_VERSION" ] && echo 'Please supply the Gold Standard version tag to pull' && show\_help

[ -z "$REPO" ] && echo 'Please supply the name of the client repository to be populated' && show\_help

MESSAGE="$REPO based on ${GOLD\_REPO}/${GOLD\_VERSION}"

declare -a cmds

cmds["${#cmds[@]}"]="git clone https://github.com/$GOLD\_USER/$GOLD\_REPO"

cmds["${#cmds[@]}"]="pushd $GOLD\_REPO"

cmds["${#cmds[@]}"]="git checkout tags/$GOLD\_VERSION -b $REPO"

cmds["${#cmds[@]}"]="curl -u \"$GOLD\_USER\" https://api.github.com/user/repos -d '{\"name\":\"$REPO\"}'"

cmds["${#cmds[@]}"]="git push --mirror https://github.com/$GOLD\_USER/$REPO.git"

cmds["${#cmds[@]}"]="popd"

cmds["${#cmds[@]}"]="rm -rf $GOLD\_REPO.git"

cmds["${#cmds[@]}"]="git clone https://github.com/$GOLD\_USER/$REPO"

cmds["${#cmds[@]}"]="pushd $REPO"

cmds["${#cmds[@]}"]="echo '# $MESSAGE' > README.md"

cmds["${#cmds[@]}"]="git add README.md"

cmds["${#cmds[@]}"]="git commit -m \"$MESSAGE\""

cmds["${#cmds[@]}"]="git push -u origin master"

cmds["${#cmds[@]}"]="popd"

[ "$VERBOSE" == "true" ] && echo "Cloning $GOLD\_USER/$GOLD\_REPO"

if [ "$DRY\_RUN" == "true" ]; then

echo 'Dry run - the following commands would be executed:'

for cmd in "${cmds[@]}"

do

echo "${cmd}"

done

else

for cmd in "${cmds[@]}"

do

eval "${cmd}"

done

fi

[ "$VERBOSE" == "true" ] && echo 'Exiting normally'

exit 0

To set up a new client repo for Mom and Pop Stores based on a version of the Gold Standard artifacts that's tagged as "gold-1.0," a consultant would enter the following command:

copy-latest-gold -g gold-1.0 -c momandpopstores

They cannot accidentally delete or corrupt anything, because the script doesn't issue any commands that would have those effects.

### Setup for Mac OSX

To use the sample script on Mac OSX, you have to obtain the GNU version of getopt and ensure it comes ahead of the default getopt in the PATH.

brew install gnu-getopt

brew link –force gnu-getopt

echo 'export PATH="/usr/local/opt/gnu-getopt/bin:$PATH"' >> ~/.bash\_profile